RL-TR-91-241
Final Technical Report
September 1991

KNOWLEDGE-BASED LOGISTICS PLANNING: ITS APPLICATION IN MANUFACTURING AND STRATEGIC PLANNING

Carnegie Mellon University

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. 6129

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-91-241 has been reviewed and is approved for publication.

APPROVED: Northrup Fowler III

NORTHRUP FOWLER III
Project Engineer

FOR THE COMMANDER: Raymond P. Urtz Jr.

RAYMOND P. URTZ, JR.
Director of Command, Control & Communications

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL(C3C) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

KNOWLEDGE-BASED LOGISTICS PLANNING: ITS APPLICATION IN MANUFACTURING AND STRATEGIC PLANNING

Mark S. Fox

Contractor: Carnegie Mellon University
Contract Number: F30602-88-C-0001
Effective Date of Contract: 30 November 1987
Contract Expiration Date: 29 November 1990
Short Title of Work: Knowledge-Based Logistics Planning
Program Code Number: 0E20
Period of Work Covered: Nov 87 - Nov 90

Principal Investigator: Mark S. Fox
Phone: (412) 268-3832

RL Project Engineer: Northrup Fowler III
Phone: (315) 330-7794

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by Northrup Fowler III, RL (C3C) Griffiss AFB NY 13441-5700 under Contract F30602-88-C-0001.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1991	Final Nov 87 - Nov 90

4. TITLE AND SUBTITLE
KNOWLEDGE-BASED LOGISTICS PLANNING: ITS APPLICATION IN MANUFACTURING AND STRATEGIC PLANNING

5. FUNDING NUMBERS
C - F30602-88-C-0001
PE - 62301E
PR - F129
TA - 00
WU - 01

6. AUTHOR(S)
Mark S. Fox

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh PA 15213

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced
Research Projects Agency
1400 Wilson Blvd
Arlington VA 22209

Rome Laboratory (C3C)
Griffiss AFB NY 13441-5700

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-91-241

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Northrup Fowler III/C3C/(315) 330-7794

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This report summarizes the activities and results of this three-year effort investigating the ideas of constrained heuristic search as applied to large scale manufacturing and transportation planning and scheduling. Topics reported on include constrained heuristic search, activity-based scheduling, distributed scheduling, constraint-directed planning, managing uncertainty and transportation planning (CDART). This effort forms the cornerstone of the constraint-directed planning activities under the joint DARPA and Rome Laboratory Knowledge-Based Planning and Scheduling Program.

14. SUBJECT TERMS
Artificial Intelligence
Planning
Constraint-based
Scheduling
Reasoning

15. NUMBER OF PAGES
198

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

List of Tables

Preface

This report describes research, performed from 1987 through 1990 in Knowledge-Based Logistics Planning as applied to Manufacturing and Strategic Planning. The research has been performed in the context of what is called the CORTES project. The report begins with an overview of the CORTES project, with the subsequent chapters elaborating individual research components.

Authorship of the chapters is as follows:

Chapter 1, Introduction: Mark Fox and Katia Sycara.

Chapter 2, Constrained Heuristic Search: Mark Fox and Norman Sadeh.

Chapter 3, Activity-Based Scheduling Norman Sadeh and Mark Fox.

Chapter 4, Distributed Scheduling: Katia Sycara, Steve Roth, Norman Sadeh, Joe Mattis, and Mark Fox.

Chapter 5, Constraint-Directed Planning: Robert Frederking and Lin Chase.

Chapter 6, Managing Uncertainty: Whay-Yu Chiang and Mark Fox.

Chapter 7, Transportation Planning (CDART): Steve Roth.

Chapter 1

The CORTES Project: A Unified Framework for Planning, Scheduling and Control

Summary

We present an overview of CORTES, an integrated framework for production planning, scheduling and control (PSC). CORTES's approach to PSC problems departs from others in the hypotheses it explores: *Generality Hypothesis*: There exists a single approach that can optimize decision making across a wide variety of PSC problems. *Flexibility Hypothesis*: The same approach can be used for both planning, predictive scheduling and reactive control. *Uncertainty Hypothesis*: In order to provide the appropriate level of precision in PSC, reasoning about uncertainty must be an integral part of the PSC approach. *Scale Hypothesis*: Large PSC problems, that contain thousands of activities, resources and constraints, must be solved in a qualitatively different manner than small PSC problems. CORTES uses Constrained Heuristic Search to make PSC decisions. In this chapter, we describe CORTES, its architecture, problem solving method, and functions including modeling, planning, scheduling, distributed scheduling, dispatching, and uncertainty management.

1.1. CORTES Philosophy

Our research explores the role of constraints in solving planning, scheduling and control (PSC) problems. It is generally believed that to efficiently construct optimizing solutions to large PSC problems, a fundamental understanding of ***problem structure*** and ***properties*** is required. It is our conjecture that knowledge of domain constraints will lead to this understanding. The goal of the CORTES project is to operationalize this conjecture.

CORTES is a distributed system for production planning, scheduling and control. CORTES is designed to be composed of an integrated set of modules distributed across many workstations and connected by a communication network. The overall architecture is shown in Figure 1-1.

CORTES represents a departure from previous approaches to solving PSC problems in the hypotheses it explores:

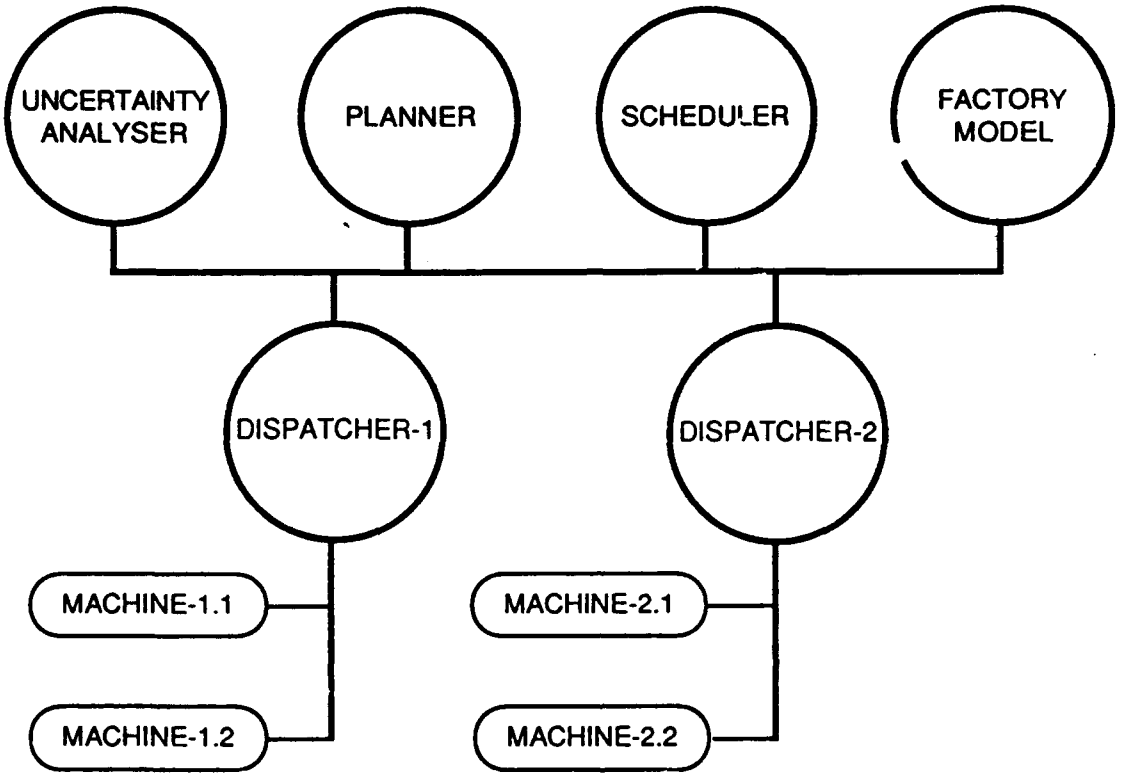


Figure 1-1: The CORTES Architecture

1. ***Generality Hypothesis:*** There exists a single approach that can optimize decision making across a wide variety of PSC problems. Previously, PSC approaches were tailored to the particular production environment, with the "common wisdom" being that there does not exist a single approach, short of enumeration, that applies to all PSC problems. We believe that there does exist a single approach that may be generally applied to PSC problems, that also provides very good results and is computationally efficient.

2. ***Flexibility Hypothesis:*** The same approach can be used for both planning, predictive scheduling and reactive control. Traditionally, planning, scheduling and control approaches have tended to be separate and unrelated in approach. For example, in actual production environments, Manufacturing Resource Planning (MRP) tends to be used for planning, scheduling approaches can range from dispatch approaches to knowledge based scheduling, and control tends to be ignored. In AI, planning algorithms tend to be generative, scheduling is constraint directed, and control is reactive.

3. ***Uncertainty Hypothesis:*** In order to provide the appropriate level of precision in PSC, reasoning about uncertainty must be an integral part of the PSC approach. Production environments contain a plethora of stochastic events that increase the uncertainty with which a schedule may be executed. For example, unexpected events such as personnel not showing up for work, machine failures, failure of resources, etc, may quickly invalidate a schedule. Consequently, PSC approachs must take a pro-active approach in mitigating the effects of uncertainty.

4. ***Scale Hypothesis:*** Large PSC problems, that contain thousands of activities, resources and constraints, must be solved in a qualitatively different manner than small PSC problems. The point is that in large PSCs, the aggregate behavior of the system be optimized, as opposed to any individual entity or job. Optimizing each decision is computationally expensive. Instead, many decisions must be made at the aggregate level using statistical summaries of underlying requirements.

CORTES is evolutionary in its approach in that it can be viewed as a continuation of the line of constraint directed scheduling systems developed at Carnegie Mellon University [Fox & Smith 84, Smith et al. 86, Fox 90]. It departs from the approach of these previous systems in its use of Constrained Heuristic Search (CHS) as its underlying problem solving paradigm [Fox et al. 89].

The remainder of this section briefly describes the four areas explored by the CORTES project.

1.2. Constrained Heuristic Search

Our approach to both planning and scheduling is based upon a problem solving paradigm we call Constrained Heuristic Search (CHS)[1] CHS views problem solving as a constraint optimization activity. CHS combines the process of constraint satisfaction (CSP) [Mackworth 87] with heuristic search (HS). CHS retains heuristic search's synthetic capabilities and extends it by adding the structural characteristics of constraint satisfaction techniques. In particular, our model adds to the definition of a problem space [Newell & Simon 76], composed of states, operators and an evaluation function, by refining a state to include:

1. **Problem Topology**: Provides a structural characterization of a problem in the form of a constraint graph.

[1]This section is a composed of excerpts from [Fox et al. 89].

2. **Problem Textures**: Provide measures of a problem topology that allows search to be focused in a way that reduces backtracking.
3. **Problem Objective**: Defines an objective function for rating alternative solutions that satisfy a goal description.

Significance: This model allows us to (1) view problem solving as constraint optimization, thus taking advantage of these techniques, (2), incorporate the synthetic capabilities of heuristic search, thus allowing the dynamic modification of the constraint model, and (3) extend constraint satisfaction to the larger class of optimization problems. In addition, a problem topology operationalizes the notion of problem structure, allowing for the development of structure-based theories of problem solving.

1.3. Scheduler

The scheduler, Micro-Boss, is an activity-based scheduler [Sadeh & Fox 90a], where the activities are the operations that must be scheduled according to a process plan that specifies a partial ordering among these operations. Each operation requires one or several resources for each of which there may be one or several alternatives. Scheduling is viewed as a constrained heuristic search problem whose solution is a schedule that satisfies the many technological, temporal, organizational, and preference constraints that are imposed both by the characteristics of the job shop itself and the environment.

The scheduler models a problem as a constraint graph, where there are two types of nodes: activities and resources. An activity is an 4-tuple defining its start time, duration, and resources it is to use. With each activity, we associate utility functions that map each possible start time and each possible resource alternatives onto a utility value (i.e. preference). These utilities [Fox 83, Sadeh & Fox 88] arise from global organizational goals such as reducing order tardiness (i.e. meeting due dates), reducing order earliness (i.e. finished good inventory), reducing order flowtime (i.e. in-process inventory), using accurate machines, performing some activities during some shifts rather than others, etc. A resource is a 3-tuple defining its total capacity, available capacity over time, and the activities that are scheduled to use it.

We distinguish between two types of constraints: activity temporal constraints and capacity constraints. The activity temporal constraints together with the order release dates and latest acceptable completion dates restrict the set of acceptable start times of each activity. The capacity constraints restrict the number of activities that a resource can be allocated to at any moment in time to the capacity of that resource. Typically the limited capacity of the resources induces interactions between orders competing for the possession of the same resource at the same time.

The schedule is built incrementally by iteratively selecting an activity and assigning a start time and resource(s) to it, propagating temporal and capacity constraints and checking for constraint violations. If constraint violations are detected the system backtracks. Search is focused via a set of *variable* and *value* ordering heuristics so as to minimize backtracking and optimize schedule quality.

The variable ordering heuristic assigns a *criticality measure* to each unscheduled activity; *the activity with the highest criticality is scheduled first.* The value ordering heuristic attempts to leave enough options open to the activities that have not yet been scheduled in order to reduce the chances of backtracking. This is done by assigning a *goodness* measure to each possible reservation of the activity to be scheduled. Both activity criticality and value goodness are composed of *texture measures.*

Significance: Scheduling at the micro level of activities provides for a finer level of control. In contrast to macro-level schedulers, it interatively re-prioritizes its decision problems at the activity level so that it does not dwell on problems of little significance and whose solution at that point may negatively impact the overall schedule.

1.4. Distributed Scheduling

We have investigated how to manage scheduling when distributed across multiple schedulers [Sycara et al. 90]. In particular, we investigated how schedulers, which possess their own resources, coordinate their decisions when they require resources possessed by others. Due to the size of the scheduling problem, we distinguish between coordination at the strategic level versus the tactical level. Our approach assumes that each scheduler develops schedules using Constrained Heuristic Search. At the strategic level, the coordination of large numbers of activities requiring resources outside of a particular scheduler is performed by communicating statistical summaries of aggregate demand textures. These demands are used to bias a scheduler's reservations so that it does not require another's scheduler's resources during a period of high demand. At the strategic level, we expect that coordination problems will be reduced but not removed. It is the role of the tactical level to negotiate resource allocations that could not be handled strategically.

Significance: We have extended the concept of Constrained Heuristic Search to distributed problem solving where each agent uses CHS as its problem solving model. Textures play four important roles in distributed search: (1) they focus the attention of an agent to globally critical decision points in its local search space, (2) they provide guidance in making a particular decision at a decision point, (3) they are good predictive measures of the impact of local decisions on system goals, and (4) they are used to model beliefs and intentions of other agents.

1.5. Planning

We investigated the integration of planning with scheduling. In previous planners, planning has been an end unto itself. Any feasible plan is considered a success, with only very inflexible criteria for plan quality, such as minimizing the total number of actions. In the context of the CORTES project, the planner produces process plans to be used by the scheduler. The quality of these plans is defined by the quality of the schedules that the scheduler can produce using them. Thus, there is a strong need for a constraint language to use in communicating with the scheduler to determine what sorts of plans would be good.

Current state-of-the-art planners are constraint-directed, domain-independent, hierarchical, nonlinear, and support replanning [Wilkins 88]. We have developed a planner that supports planning at different levels of abstraction, and the re-use of plans in support of reactive planning. We are still investigating the use of textures to guide the planning process.

Significance: If planners are to be used in "real world" situations, they must be aware both of the complexity of the domain and the constraints the domain imposes. Experience in scheduling applications has shown that the development of plans without consideration of time and resources constraints leads to significants amount of replanning. By integrating constrained heuristic search techniques into planning, feasible plans can be generated with less search.

1.6. Uncertainty Analyzer

We focus on the relationship between domain uncertainty and temporal precision in the development and execution of schedules. It is obvious that as uncertainty increases in a domain, it is less likely that a predictive schedule will be implemented successfully and it is more likely that there will be a greater reliance on a reactive scheduler to generate an appropriate response [Fox & Smith 84, Ow et. al. 88]. Consequently, spending more time on developing precise schedules that further optimize a set of measures, such as reducing work-in-process or tardiness, may be unnecessary if the temporal granularity of the impact of uncertainty exceeds the granularity of the optimization of the schedule. But given the necessity of developing predictive schedules, in order to plan resource purchases, allocations and releases, the question arises as to how precise should temporal decisions be, and the nature of the flexibility that a reactive scheduler should be afforded in responding to stochastic events. The domain in which we explore these issues is manufacturing scheduling.

CORTES manages uncertainty in three stages. In the first stage, the Uncertainty Analysis

module monitors and records the stochastic events. It develops over time a model of the sources and characteristics of uncertainty. Once a valid model is constructed, the Uncertainty Analysis module passes the information to the Scheduler. In the second stage, the scheduler uses the uncertainty models to reduce the precision of its schedules. Precision can be reduced by increasing the durations of activities, overlapping activity temporal intervals, or assigning activities to resource aggregates rather than to specific resources. In the third stage, the Dispatcher control module, is able to react more flexibly to stochastic events by taking advantage of the imprecision inserted in the schedule by the scheduler; it can start an activity earlier or later or assign an activity to another resource in an aggregate (i.e., work center). The Dispatcher's task is to dispatch jobs to machines and monitor machine and job execution status. The Dispatcher notes deviations from the schedule and resource unavailability and communicates this information to the Scheduler, Uncertainty Analyzer and factory floor.

The CORTES uncertainty analyzer represents uncertainty in terms of fuzzy logic [Zadeh 65, Kaufmann & Gupta 85, Prade 79]. The present version [Chiang & Fox 90] focuses on uncertainty concerning machine failures. The mean time between failure and mean duration of the failure are assumed known. It is also assumed that once a machine is fixed after a failure, processing resumes at the point of interruption with no rework necessary. In other words, machine failure causes a variation in processing time only and not in scheduling order. The time between machine failures and the failure duration are used to express uncertainty in processing time. Instead of being random variables of known distribution, the duration of failure and time between failures may be only approximately known. This approximate information on the processing time bounds is expressed in terms of fuzzy numbers of *Type-1*, where a real number that is approximately known is expressed as a confidence interval of upper and lower bounds. Fuzzy bound values may be the result of subjectively known processing characteristics described by a shop operator, or of known distributions described by shop statistics. An extension of type-1 fuzzy representation of uncertainty in operation duration is *Type-2* representation where the lower and upper bounds of a confidence interval, instead of being ordinary numbers are fuzzy numbers that themselves have intervals of confidence. Uncertainty bounds work in a similar manner as earliest start time/latest start time and earliest finish/latest finish time. The bounds can be viewed as slack to protect against uncertainty. The mean processing time is reserved for the operation and the slack time is reserved for protection against uncertainty. Once an operation is ready for processing, a dispatcher should follow the schedule within the prescribed bounds.

Significance: The impact of uncertainty on the robustness of plans and schedules has only recently become of interest, and primarily in the robot planning world whose characteristics differ from that of manufacturing environments. This research demonstrates

how statistical knowledge of failures can be used to modify activity reservations so that costs of work-in-process, tardiness and facility idleness are minimized.

Chapter 2
Constrained Heuristic Search

Summary

We propose a model of problem solving that provides both structure and focus to search. The model achieves this by combining constraint satisfaction with heuristic search. We introduce the concepts of topology and texture to characterize problem structure and areas to focus attention respectively. The resulting model reduces search complexity and provides a more principled explanation of the nature and power of heuristics in problem solving.

2.1. Introduction

We propose a model of problem solving that provides both structure and focus to search in the problem space. The model achieves this by combining the process of constraint satisfaction (CSP) with heuristic search (HS). The resulting model both reduces search complexity and provides a explanation of the nature and power of heuristics in problem solving.

Our problem solving model, called *Constrained Heuristic Search* (CHS), retains heuristic search's synthetic capabilities and extends it by adding the structural characteristics of constraint satisfaction techniques. In particular, our model adds to the definition of a problem space, composed of states, operators and an evaluation function, by refining a state to include:

1. **Problem Topology**: Provides a structural characterization of a problem.
2. **Problem Textures**: Provide measures of a problem topology that allows search to be focused in a way that reduces backtracking.
3. **Problem Objective**: Defines a means for rating alternative solutions.

This model allows us to (1) view problem solving as constraint satisfaction, thus taking advantage of these techniques, (2), incorporate the synthetic capabilities of heuristic search, thus allowing the dynamic modification of the constraint model, and (3) extend constraint satisfaction to the larger class of optimization problems. In the following, we define the scheduling problem to be used as an example throughout the chapter, followed by a definition of problem topology, textures, objectives and the CHS search process.

2.2. Factory Scheduling Example

[2] Factory scheduling involves the assignment of start times and resources to a set of activities. Each activity belongs to an order (i.e. job). Activities within the same order are subject to precedence constraints as specified by a process plan. Additionally no two activities are allowed to use the same resource at the same time (we assume resources of unary capacity). Each order has a release date and a latest acceptable completion date (which may be later than the due date), that can be used to determine an earliest start time and a latest start time for each activity in the order. Additionally each activity may require one or several resources, for each of which there may be several alternatives. For each activity, utility functions map each possible start time and each possible resource alternative onto a utility value (preference). The sum of these utilities over all the activities to be scheduled defines an objective function to be maximized. These utilities [Fox 87, Sadeh & Fox 88] arise from organizational goals such as reducing order tardiness, reducing order flowtime, using accurate machines, performing some activities during a specific shift, etc.

2.3. Problem Topology

It is our conjecture that an understanding of the structure of a problem will lead to more effective problem solving methods. Therefore, our goal is to formalize the concept of problem structure.

In an attempt to distinguish the problems to which AI was being applied, from problems which conventional algorithms were being applied, the notion of "well-structuredness" arose. Simon defines a well-structured problem as follows [Simon 73]:

1. There is a definite criterion for testing any proposed solution, and a mechanizable process for applying the criterion.
2. There is at least one problem space in which can be represented the initial problem state, the goal state, and all other states that may be reached, or considered, in the course of attempting a solution to the problem.
3. Attainable state changes (legal moves) can be represented in a problem space, as transitions from given states to the states directly attainable from them. But considerable moves, whether legal or not, can also be represented -- that is, all transitions from one considerable state to another.
4. Any knowledge that the problem solver can acquire about the problem can be represented in one or more problem spaces.
5. If the actual problem involves acting upon the external world, then the definition of state changes and of the effects upon the state of applying any operator reflect with complete accuracy in one or more problem spaces the laws (laws of nature) that govern the external world.

---

[2]Excerpted from [Fox et al. 89].

6. All of these conditions hold in the strong sense that the basic processes postulated require only practicable amounts of computation, and the
7. information postulated is effectively available to the processes -- i.e., available with the help of only practicable amounts of search.

Most of the requirements focus on the feasibility of operationalizing, in a computational sense, the means for solving the problem, using the problem space model. But the sixth requirement focuses on the "strength" of the method. Newell defines a "weak" method as one that makes weak information demands and gives weak results, such as generate and test and hill climbing [Newell 73]. Using the big switch approach to problem solving where alternative methods exists for solving a class of problems, weak methods are the methods of last resort - that is, when other "stronger" methods fail we can use weak methods. Simon's sixth requirement reiterates the essence of Newell's definition of an ill-structured problem [Newell 69]: A problem is ill structured if there only exists weak methods to solve it; problem solving performance is the key concern.

Other definitions of a problem's structure have been proposed. Eastman defines a problem as being ill-structured if problem formulation proceeds concurrently with its solution, as typically found in design problems [Eastman 69]. Reitman defines a problem as being ill-structured when both the problem and goals are well defined by the method of solving it are not [Reitman 65]. None of these definitions provide insight into *how* to solve a problem more effectively.

During the last fifteen years, research in AI has begun to exploit various structural characteristics of problems. The success of expert systems relies upon the recognition and exploitation of recurring patterns in a problem [Waterman & Hayes-Roth 78]. More recently, the SOAR model [Laird et al. 87] and its interpretation in the RIME methodology [van de brug et al. 86] provides a structure for rules themselves, dividing them into rules for proposing, prioritizing, selecting and implementing operators.

Within the heuristic search model, a variety of techniques for structuring problems have been investigated. ABSTRIPS [Sacerdoti 74] demonstrated how hierarchical reformulation of the problem via the ranking and omission of variables reduces search complexity. Hearsay-II utilized data aggregation to reduce search complexity in a domain with high data uncertainty [Erman et al. 80]. MOLGEN utilized operator aggregation to restrict the set of operators used to construct detailed plans [Stefik 81]. ISIS demonstrated how hierarchical reformulation via omission of constraints reduces search complexity [Fox 87]. In each case, the systems provide an example of how the structuring of a problem can occur, but they do not provide a theory of structuring. What should be the basis of a theory of problem structure?

Problem structure has long been an interest in Operations Research. In particular,

mathematical programming views problem solving as finding a solution to a set of ***constraints*** that optimizes an objective function. The success of linear programming stems from the recognition that a problem can be defined by a set of linear constraints that define a subspace in an n-dimensional euclidean space, and that the optimal solution, as defined by an objective function can be found at one of the vertices of delineated subspace. The same is true of nonlinear programming where the convexity of the solution space implies that hill climbing can be used to find an optimal solution.

We have seen the constraint perspective of problem solving reappear in the last fifteen years, within AI, in form of Constraint Satisfaction Problems (CSP). Constraint satisfaction techniques, as described in [Mackworth 77, Haralick & Elliott 80, Freuder 82, Dechter & Pearl 87], approach problem solving by constructing a constraint graph where nodes are variables with discrete domains and arcs are n-ary constraints. Problem solving is performed by sequentially choosing a variable and a value to assign to it that satisfies all constraints incident upon it. Backtracking occurs when an assignment cannot be found. Research has gone into methods for structuring the network so that the amount of backtracking can be reduced. Arc-consistency is one such technique that achieves local consistency between groups of variables via the elimination of incompatible values [Montanari 74, Mackworth 77, Davis 87].

The generality of the CSP model of problem solving, when extended to include continuous variables such as time and space, has been demonstrated across problems such as spatial planning [Baykan & Fox 87], scheduling [Dincbas et al. 88, Elleby et al. 88, Sadeh & Fox 88], diagnosis [Davis & Hamscher 88] and truth maintenance.

We can now return to the question posed earlier: What should be the basis of a theory of problem structure? The experience of both OR and CSP provides evidence that many problems can be adequately modeled by a constraint graph, and that the structure of these graphs can have significant impact on how to solve a problem. Therefore we adopt the view that a problem's structure is defined by its constraint graph.

We define ***problem topology*** as a graph G, composed of variables V and constraints C. Each variable may be a vector of variables whose domains may be finite/infinite and continuous/discrete. Constraints are n-ary predicates over variables. We distinguish between two types of problem topologies:

> **Definition 1:** A *completely structured problem* is one in which all non-redundant vertices and edges are known a priori.

This is true of all CSP formulations.

> **Definition 2:** A *partially structured problem* is one in which not all non-redundant vertices and edges are known prior to problem solving.

This definition tends to be true of problems in which synthesis is performed resulting in new variables and constraints (e.g. the generation of new subgoals during the planning process).

We view the scheduling problem as an optimization version of the CHS model, where each activity is an aggregate variable whose values are reservations. A reservation consists of a start time and a set of resources to be allocated to the activity. Each activity constitutes a variable vertex in the problem topology. Activity precedence constraints are binary constraints represented by constraint vertices connected to two activity variable vertices. A capacity constraint vertex is associated to each physical resource of the domain and connected to all the variable vertices representing activities that can possibly use the resource. Each capacity constraint ensures that the corresponding resource will not be allocated to more than one activity at any given time. Accordingly we distinguish between two types of constraint interactions:

- the *intra-order* interactions defined by the precedence constraint vertices between activities belonging to a same order, and
- the *inter-order* interactions induced by the capacity constraint vertices between activities contending for a same resource.

Both types of interactions contribute to the contention of each activity.

Features of the problem topology are the types of variables and constraints (and their associated propagation algorithms). Davis [Davis 87] mentions two classes of what we view as topological features, namely the types of values the domain of a variable may contain, such as variables whose domains are discrete and finite (label and value inference), are intervals, have belief for each member (relaxation labeling), and are expressions (expression inference). The second class of features focus on the types of constraints, such as constraints that are unary predicates, order relations, bounded differences (e.g. $x-y \geq c$), linear equations with unit (i.e. -1, 0, 1) coefficients, linear equalities and inequalities with arbitrary coefficients, boolean combinations of constraints, algebraic equations, and transcendental equations. Additionally, domains may or may not have preferences for values (e.g. preferences for due dates of a job).

Informal notions of problem structure can be formalized by a problem's topology. Where and how to search can be guided by structure. For example, problem decomposition can be viewed as a decomposition of a topology into sub-graphs. Means for determining a decomposition may vary according to a problem's constraints and objectives [Alexander 65, Alexander 68, Courtois 77]. Situations in which search is efficient, such as backtrack free search in width 1 graphs, can also be identified [Freuder 82].

Problem situations can be identified by patterns in the topology. For example, difficult constraint situations can be identified by "knots" in the topology [Krishnan et al. 90].

Problems can be simplified by topology manipulation. Problem reformulation, by means of relaxation or abstraction can be explicitly defined in terms of topology. Relaxation is a process by which constraints are relaxed to admit more solutions. Abstraction can be defined in terms of variable or constraint aggregation or omission.

Another type of reformulation of the problem topology is the creation of a "contention graph" [Fox & Sadeh 90]. Consider the factory scheduling problem where many operations are contending for a small set of machines. The allocation of these machines over time must be optimized. This is equivalent to having a set of variables, with small discrete domains, each competing for the assignment of the same value but linked by a disequality constraint. A contention graph replaces disequality constraints by a node for each value under contention, and links these "value nodes" to the variables contending for it by a demand constraint. We can now use these value nodes to measure the amount of contention for the value by variables. Contention is an important metric for variable and value ordering [Fox et al. 89].

What value do we derive from viewing a problem space state as a constraint graph? First, we have provided a more refined definition of a problem space state thereby reducing the looseness of its definition and allowing the definition of general measures of problem structure, i.e., textures. Second, properties can be proved about the nature of the problem, e.g., width-1 constraint networks that are arc consistent are backtrack free. Third, the process of problem reformulation can be viewed as transformations of problem topological primitives. A possible negative, is that the number of problem types that can be represented in the form of a constraint graph is limited. But this set is growing larger; in the factory scheduling example, we have shown how the representation can be extended to handle optimization [Sadeh & Fox 90a]. By adding the power of heuristic search, we believe that we can apply the model to a broader class of problems.

2.4. Problem Textures

Evaluation functions play a prominent role in search; whether to identify the node to expand next in best first search, or to recognize an island to extend in opportunistic search. The problem is that most evaluation functions are "hand molded" for each problem, thereby limiting their reusability. The question is whether there are measurements of the problem topology that give rise to powerful heuristics and are problem invariant?

In CHS, for search to be well focused, there must be measures of the topology that differentiate one subgraph from another, and these measures must be related to the goals of the problem. We have identified and are experimenting with six such measures that we call problem *textures* [Sadeh & Fox 88, Fox et al. 89]:

- **Value Contention:** Degree to which variables are contending for the same value.

  In the factory scheduling domain, measuring the amount of contention there exists for resources is important in identifying bottlenecks. It is the activities associated with the bottleneck resource that are assigned first [Fox & Sadeh 90].

- **Value Conflict:** Degree to which a variable's assigned value is in conflict with existing constraints.

In repairing schedules, focusing on activities that participate in the greatest number of conflicts reduces search [Minton et al. 90].

- **Value Reliance:** Degree to which a variable relies upon the availability of a particular value.

  Once contention is used to identify a resource and the activities contending for it, selecting an activity depends upon the degree to which the activity relies upon having the resource.

- **Value Goodness:** Probability that the value leads to the best solution.

  Once a resource is selected for an activity, deciding when the resource should be assigned for use depends upon how disruptive the assignment would be to subsequent assignments [Sadeh & Fox 90a].

- **Constraint Tightness:** Degree to which a constraint reduces the number of solutions.

  If the scheduling problem is large and too complex to solve directly, then solving a simpler version may provide guidance in solving the original. Reformulating the problem can be accomplished by omitting "loose" constraints.

- **Variable Tightness:** Degree to which a solution to the problem is constrained by a particular variable.

  Variable tightness can be used to decide which variables to aggregate in a reformulation.

These textures generalize the notion of constraint satisfiability or looseness defined by [Nadel 86] and apply to both CHSs (and CSPs) with discrete and continuous variables. There exist many ways in which to perform these measurements. Textures may sometimes be evaluated analytically [Sadeh & Fox 88]. Such techniques may however be very costly. In general, for a given CHS, some textures are easier to approximate than others, and some are also more useful than others. Usually the texture measures that contain the most information are also the ones that are the most difficult to evaluate. Hence there is a tradeoff. Each domain may have its own approximation for a texture measure.

2.5. Problem Objectives

Many problems, such as design and scheduling, require the optimization of one or more objectives in addition to the satisfaction of a set of constraints. In linear programming, an objective function is used to evaluate each vertex visited, and in heuristic search an evaluation function is used to rate each state. In either case, objectives tend to be combinations of more primitive statistics. For example, in scheduling, the objective is a weighted combination of tardiness and flowtime. It has long been understood, that by moving more of the evaluation knowledge into the state generator, a more effective search can be performed. An analogous situation exists in CHS.

With CHS, we can view each objective as being a constraint with associated utilities. For

example, a due date in the scheduling domain is a temporal constraint that specifies a set of acceptable dates for the completion of an activity. We can extend the representation to include preferences in the form of utilities for each due date [Fox 87]. This constraint, which is represented directly in the problem topology, represents a local preference. When selecting a variable and assigning a value to it, optimizing any local constraints is straight forward. But in order to optimize our decision making, each local decision in the constraint graph should be globally optimal. This can be achieved by constraint propagation. In particular, temporal preferences can be propagated such that local preferences can be transformed to represent preferences more globally optimal [Sadeh & Fox 88]. The power of representing a problem as a constraint graph enables the determination, by propagation, of how individual decisions will impact each via constraints.

2.6. CHS Problem Solving Process

The CHS model of problem solving is a combination of constraint satisfaction and heuristic search. Search is performed in the problem space where each state is defined by a problem topology. For most problems, the problem topology is only partially complete. Therefore, a goal of the search process is to acquire additional topology or modify existing topology. Optimization of the decision process occurs naturally as constraint propagation transform local preferences into more global preferences.

The problem solving model we propose contains the following elements:

- An initial state is defined composed of a problem topology,
- Constraint propagation is performed within the state,
- Texture measures are evaluated for the state's topology,
- Operators are matched against the state's topology, and
- A variable node/operator pair is selected and the operator is applied.

The application of an operator results in either adding structure to the topology, further restricting the domain of a variable, or reformulating the problem (e.g., relaxation).

In our scheduling domain example[3]: Search begins with a single state where all activities still have to be scheduled and all resources are available. Scheduling an activity in a state with a reservation results in the creation of a new search state where new constraints resulting from the assignment of the reservation to to the activity are propagated. The propagation consists in updating the domain of start times and resources that remain possible for each unscheduled activity [Sadeh & Fox 88]. If an inconsistency is detected the system backtracks. Next the scheduler computes a contention/reliance measure for each

[3]Excerpt from [Fox et al. 89]

unscheduled activity. The activity with the highest contention/reliance is selected to be scheduled next. A value goodness measure is computed to select the first reservation to be tried for that activity (among the reservations that are still possible). The process goes on until all activities have been scheduled or until all search states have been visited.

2.7. Conclusion

The creation of general models for problem solving has been of continuing interest to Artificial Intelligence researchers. The process is evolutionary, elaborating and/or creating new search methods and richer representations of knowledge. The SOAR architecture, for example, combines both the problem space and production system models and extends them with universal subgoaling and chunking, thus achieving a model with powerful learning capabilities. But within this model, there are two aspects of the problem space that remain ill-defined: the notion of structure and means of focusing attention within a structure. Our model, Constrained Heuristic Search, extends the problem space model in these directions. Problem topology provides a definition of structure in the form of a constraint graph. Problem textures provide a graph theoretic definition of the complexity and importance of decisions within a topology. Problem objectives define an objective function that after constraint propagation provide indicate the global optimality of a local decision. Together they enable the problem solver to direct search more economically towards a higher quality solution.

The model has been demonstrated in four domains: spatial planning [Baykan & Fox 90], factory scheduling [Fox & Sycara 90, Sadeh & Fox 90a], transportation planning [Camden et al. 90], and resource configuration [Dunmire et al. 90]. In spatial planning, we demonstrated that CHS is more efficient in finding solutions than other comparable systems. In factory scheduling, we generalized constraint graphs to account for preferential temporal constraints, making it possible to represent the general job shop scheduling problem for the first time. Texture measures, based upon these preferences, enabled the scheduler to opportunistically select the next best decision to make. They also provided an explanation of the power of domain heuristics like bottleneck analysis.

Chapter 3
Activity-Based Scheduling

Summary

Recent research in factory scheduling has demonstrated the benefits of building schedules by first focusing on the sequencing of bottleneck machines. Typically, approaches suggested in the literature schedule an entire bottleneck machine, before checking if a new bottleneck has appeared. If a new bottleneck is detected, the system schedules all the operations requiring that new bottleneck, otherwise job schedules are completed one by one around the bottleneck schedules. In these schedulers, problem solving is opportunistic in the selection of *entire* bottleneck machines, which are scheduled one by one. For this reason, these schedulers are referred to as *macro-opportunistic* schedulers. In this chapter, it is argued that bottlenecks do not necessarily span over the entire scheduling horizon. Furthermore they typically shift as scheduling decisions are made. A new *micro-opportunistic* approach to factory scheduling has been implemented in the context of the MICRO-BOSS factory scheduling system. This approach allows for continuously monitoring the evolution of bottlenecks during the construction of the schedule. As soon as MICRO-BOSS detects that the main bottleneck has moved, it shifts its attention to the new bottleneck. Sequencing decisions at the bottleneck are made by explicitly trading tardiness and inventory costs. A large scale computational study is reported that compares MICRO-BOSS against the Weighted Shortest Processing Time (WSPT) rule, the Earliest Due Date (EDD) rule, the Slack per Remaining Processing Time rule (S/RPT), the Weighted Cost Over Time (Weighted COVERT) rule, and two coarser opportunistic schedulers. The results indicate that the micro-opportunistic approach allows for important reductions in inventory while consistently maintaining tardiness at a level comparable to the one obtained by the best priority rules. They also indicate that the performance of opportunistic schedulers degrades as their granularity increases, thereby demonstrating the superiority of the micro-opportunistic approach to scheduling.

3.1. Introduction

In an economy where competition continuously intensifies, the need for cost-efficient production control is becoming more critical every day. Current production control techniques have largely failed to provide such cost-efficient solutions because of their inability to account for the diversity of constraints and preferences typically encountered in the production control domain. Numerous techniques have been developed that focus on meeting due dates without paying attention to inventory costs, or attempt to maximize machine utilization without taking care of meeting due dates. This chapter presents an approach to scheduling that can flexibly account for a variety of costs including tardiness costs, in-process inventory costs, and finished-goods inventory costs. These costs are used to continuously update demand profiles that reflect contention between unscheduled jobs for the allocation of machines in function of time. By closely monitoring the evolution of bottlenecks during the construction of the schedule, and focusing on resolving tradeoffs at these bottlenecks, this scheduling approach has allowed for impressive reductions in inventory while maintaining tardiness at a very low level.

The job shop scheduling problem requires scheduling a set of jobs on a finite set of resources (e.g. machines, human operators, etc.). Each job is a request for the scheduling of a set of operations according to a process routing that specifies a partial ordering among these operations, along with their resource requirements. Operations are atomic: once started they cannot be interrupted. This chapter is concerned with the design of a factory scheduler that builds a detailed schedule for the jobs to be produced over a planning horizon that ranges from a couple of days to a couple of weeks. The jobs to be scheduled are provided by a master-scheduler [Silver & Peterson 85] along with due dates, earliest acceptable release dates, marginal tardiness costs, marginal in-process inventory costs (e.g. interests on raw material costs, marginal direct holding costs, interests on processing costs, etc.), and marginal finished-goods inventory costs.

Job shop scheduling belongs to the class of NP-complete problems [Garey & Johnson 79]. At the exception of a couple of one-, two-, and three- machine job shop scheduling problems, for which there exist efficient algorithms [Rinnooy Kan 76], all attempts to guarantee an optimal solution have failed. Instead job shop scheduling problems have traditionally been solved using priority dispatch rules [Baker 74, Panwalkar & Iskander 77, French 82]. These are local decision rules of the greedy type that build schedules via a forward simulation of the shop. Because these rules lack a global view of the shop, they usually build up inventory in front of bottleneck machines.

More recently, with the advent of more powerful computers, a couple of more sophisticated scheduling methods have been developed [Goldratt 80, Ow 85, Adams et al. 88, Ow & Smith

88, Morton et al. 88]. The OPT/SERVE scheduling technique, by far the most publicized of these techniques, emphasized among other things the need to distinguish between bottleneck and non-bottleneck machines [Jacobs 84, Fox 87]. In OPT/SERVE, the SERVE module produces an initial infinite capacity schedule by working backwards from the job due dates. This initial schedule helps detect potential bottlenecks. The OPT module is then called upon to generate a forward finite capacity schedule that optimizes the utilization of the bottlenecks. The resulting bottleneck schedules are passed back to the SERVE module, which schedules the non-bottleneck operations while trying to minimize inventory. The distinction between bottleneck and non-bottleneck machines was pushed one step further in the OPIS system [Smith et al. 86, Ow & Smith 88], where it was recognized that new bottlenecks can appear during the construction of the schedule. The OPIS scheduler combines two scheduling perspectives: a resource-centered perspective for scheduling bottleneck resources, and a job-centered perspective to schedule non-bottleneck operations on a job by job basis [Fox 83]. Rather than relying on its initial bottleneck analysis, OPIS typically repeats this analysis each time a resource or a job has been scheduled. This ability to detect the emergence of new bottlenecks during the construction of the schedule has been termed *opportunistic scheduling* [Ow & Smith 88]. However, the opportunism in this approach remains limited in the sense that it typically requires scheduling an entire bottleneck before being able to switch to another one. For this reason, such scheduling techniques should in fact be called *macro-opportunistic*.

In reality, bottlenecks do not necessarily span over the entire scheduling horizon. Moreover they tend to shift before being entirely scheduled. A more flexible approach would allow to quit scheduling a resource as soon as another resource is identified as being more constraining.[4] In fact, in the presence of multiple bottlenecks, one can imagine a technique that constantly shifts attention from one bottleneck to another rather than focusing on the optimization of a single bottleneck at the expense of others. For these reasons, it seems desirable to investigate a more flexible approach to scheduling, or a *micro-opportunistic* approach, in which the evolution of bottlenecks is continuously monitored during the construction of the schedule, and the problem solving effort constantly redirected towards the most serious bottleneck. In its simplest form, this micro-opportunistic approach results in an *operation-centered* view of scheduling, in which each operation is considered an independent decision point and can be scheduled without requiring that other operations using the same resource or belonging to the same job be scheduled at the same time.

Section 2 of this chapter gives a formal definition of the job shop scheduling problem assumed in this study. Section 3 describes the **MICRO-BOSS** (MICRO-BOttleneck

[4] [Adams et al. 88] describes an alternative approach in which resources can be resequenced to adjust for resource schedules built further down the road. This approach has been very successful at minimizing makespan. Attempts to generalize the procedure to account for due dates seem to have been less successful so far [Serafini et al. 88].

Scheduling System) factory scheduler[5], concentrating on the heuristics to decide which operation to schedule next and which reservation to assign to that operation. Section 4 describes an empirical study that compares MICRO-BOSS against several dispatch schedulers and two coarser opportunistic schedulers. Conclusions are presented in section 5.

3.2. The Job Shop Model

Formally, we have to schedule a set of jobs $J=\{j_1,...j_n\}$ on a set of physical resources $RES=\{R_1,...,R_m\}$. Each job j_l consists of a set of operations $O^l=\{O^l_1,...,O^l_{n_l}\}$ to be scheduled according to a process routing that specifies a partial ordering among these operations (e.g. O^l_i BEFORE O^l_j). We restrict our attention to in-tree process routings, in which operations can have several direct predecessors but at most one direct successor (i.e. assembly-type of process routings). Figure 3-1 displays two examples of in-tree process routings.

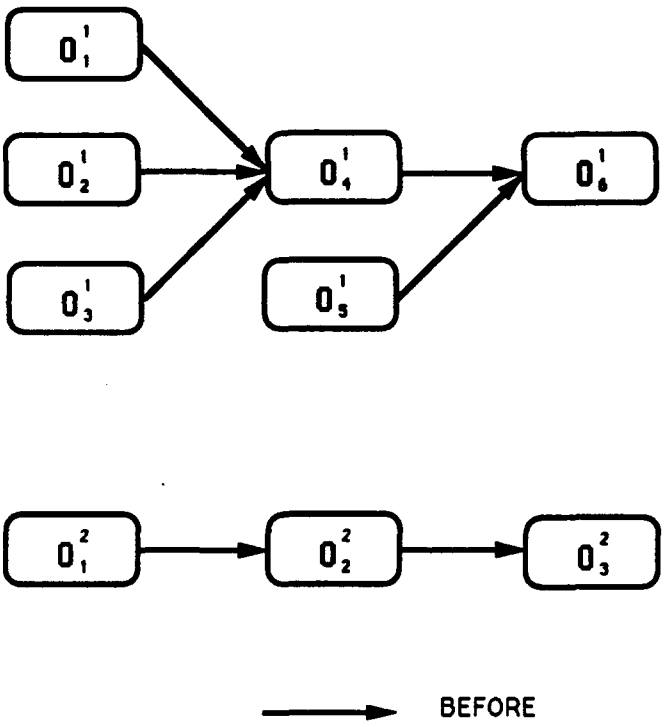


Figure 3-1: Two examples of in-tree process routing.

Additionally a job j_l has an earliest acceptable release date erd_l, a due-date dd_l, and a latest acceptable completion date lcd_l. It is assumed that for each job j_l, $lcd_l \geq dd_l \geq erd_l$. All jobs

[5]MICRO-BOSS also serves as the scheduling module of the CORTES decentralized production control system developed at the Center for Integrated Manufacturing Decision Systems at Carnegie Mellon University [Sycara et al. 91].

need to be scheduled between their earliest acceptable release date and latest acceptable completion date. The earliest acceptable release date may correspond to the earliest possible arrival date of raw materials or to a rough release date provided by a master scheduling module. It is assumed that the actual release date will be determined by the schedule to be constructed. The latest acceptable completion date may correspond to a date after which the customer will refuse delivery. If such a date does not actually exist, it can always be chosen far enough in the future so that it is no longer a constraint[6].

Each operation O_i^l has a fixed duration du_i^l and a start time st_i^l (to be determined) whose domain of possible values is delimited by an earliest start time, est_i^l, and a latest start time, lst_i^l (initially derived from the job's earliest acceptable release date erd_l and latest acceptable completion date lcd_l). In order to be successfully executed, each operation O_i^l requires a resource R_i^l (e.g. $R_i^l = R_1$, a milling machine).

VARIABLES:

The variables of the problem are the start times st_i^l.

CONSTRAINTS:

The constraints of the problem are of two types:

1. The **precedence constraints** defined by the process routings translate into linear inequalities of the type: $st_i^l + du_i^l \leq st_j^l$ (i.e. O_i^l BEFORE O_j^l).
2. The **capacity constraints** that restrict the use of each resource to only one operation at a time translate into disjunctive constraints of the type: $R_i^k \neq R_j^k \vee st_i^k + du_i^k \leq st_j^l \vee st_j^l + du_j^l \leq st_i^k$. These constraints simply express that, unless they use different resources, two operations O_i^k and O_j^l cannot overlap[7].

As mentioned earlier, each job has to be performed between its earliest acceptable release date and latest acceptable completion date. Time is assumed discrete. Operation start times and end times can only take integer values.

COSTS:

Each job j_l has:

---

[6]Notice that, although this chapter focuses on solving problems that admit at least one solution (i.e. it is possible to schedule all jobs between their earliest acceptable release date and their latest acceptable completion date), the scheduling model that has been defined allows for infeasible problems as well. Issues pertaining to the detection of infeasible problems are only briefly discussed in this chapter. A more detailed discussion can be found in [Sadeh 91].

[7]These constraints would have to be generalized to deal with resources of capacity greater than one.

• a **marginal tardiness cost**, $tard_l$: the cost incurred for each unit of time that the job is tardy (i.e. completes past its due date). The total tardiness cost of job j_l, in a given schedule, is:

$$TARD_l = tard_l \times Max(0, C_l - dd_l) \tag{3.1}$$

where $C_l = st^l_{n_l} + du^l_{n_l}$ is the completion date of job j_l in that schedule, assuming that $O^l_{n_l}$ is the last operation in job j_l.

• **marginal in-process and finished-goods inventory costs**: In our model, each operation O^l_i can incrementally introduce its own non-negative marginal inventory cost, inv^l_i. Typically the first operation in a job introduces marginal inventory costs that correspond to interests on the costs of raw materials, interests on processing costs (for that first operation), and marginal holding costs. Downstream operations[8] introduce additional marginal inventory costs such as interests on processing costs or interests on the costs of additional raw materials required by these operations. The total inventory cost for a job j_l, in a given schedule, is:

$$INV_l = \sum_{i=1}^{n_l} inv^l_i \times [Max(C_l, dd_l) - st^l_i] \tag{3.2}$$

Notice that, for the sake of simplicity, inventory costs are always counted from the start time of the operation that introduces them[9].

The total cost of a schedule is obtained by summing the cost of each job schedule:

$$Schedule\ Cost = \sum_{l=1}^{n} (TARD_l + INV_l)$$

The goal of the scheduler is to produce a feasible schedule that reduces this total cost as much as possible.

EXAMPLE:

---

[8]An operation O^k_i is said to be downstream (upstream) of another operation O^k_j within the same job if O^k_i is a direct or indirect successor (predecessor) of O^k_j in that job, as defined by its process routing.

[9]Some costs such as direct holding costs are typically incurred after the operation is completed. However, since each operation is assumed to have a fixed duration (processing time), this simplification is equivalent to adding a *fixed* cost to the total schedule cost. In other words, the difference between the costs of two schedules is not affected by this simplification.

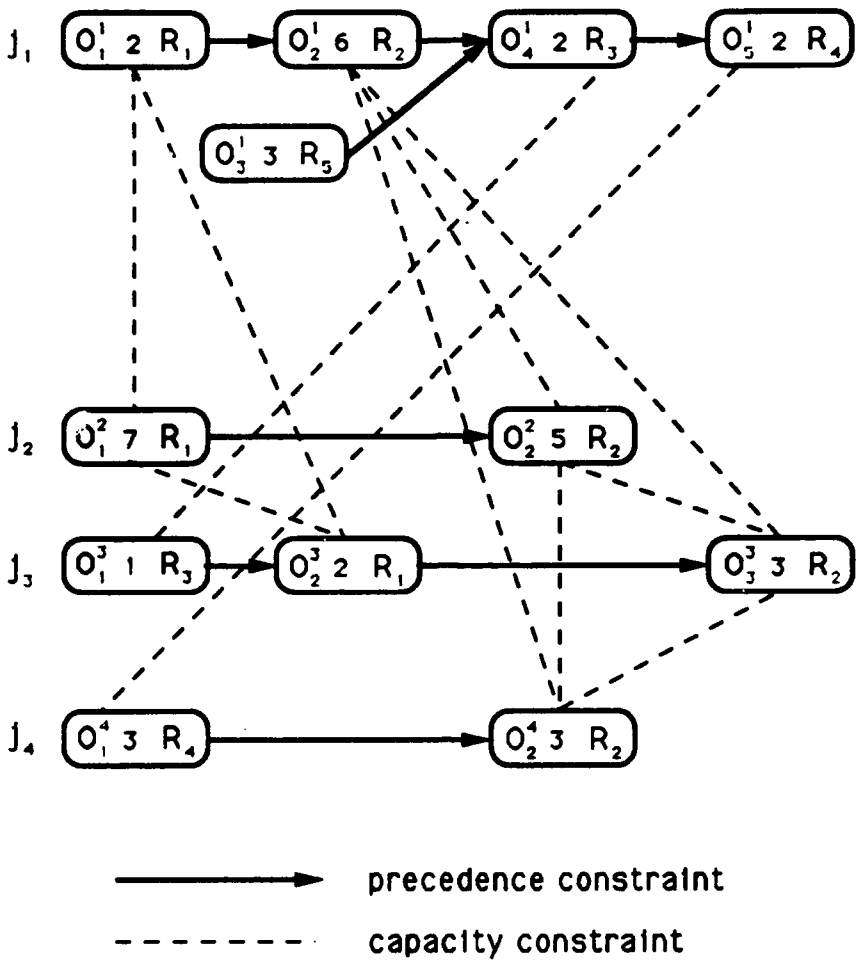


Figure 3-2: A simple job shop problem with 4 jobs. Each node is labeled by the operation that it represents, its duration, and the resource that it requires.

Figure 3-2 depicts a small job shop problem with 4 jobs. Each square box represents an operation. Each box is labeled with the name of the operation that it represents (e.g. O_1^1), the duration of that operation (e.g. $du_1^1 = 2$), and its resource requirement (e.g. $R_1^1 = R_1$). The arrows represent precedence constraints. For instance, job j_1 requires 5 operations O_1^1, O_2^1,..., O_5^1. O_1^1 has to be performed before O_2^1, O_2^1 before O_4^1, etc. The other arcs in the graph represent capacity constraints. There is a capacity constraint between each pair of operations that require the same resource (i.e. there is a clique of capacity constraints for each resource). For instance, the clique for R_2 involves operations O_2^1, O_2^2, O_3^3 and O_2^4. Notice that R_2 is the only resource required by four operations (one from each job). Notice also that, in three out of four jobs (namely j_1 j_3 , and j_4), the operation requiring R_2 is one of the job's longest operations. We can consequently expect R_2 to be the main bottleneck of the problem. We will see that, to some extent, resource R_1 constitutes a secondary bottleneck.

The earliest acceptable release dates, due dates, and latest acceptable completion dates of the jobs are provided in the table below along with the marginal tardiness and inventory costs of these jobs.

Earliest acceptable release dates, due dates, latest accept. completion dates, and costs									
Job j_l	erd_l	dd_l	lcd_l	$tard_l$	inv_1^l	inv_2^l	inv_3^l	inv_4^l	inv_5^l
j_1	0	12	20	20	2	1	2	0	0
j_2	0	14	20	20	5	0	-	-	-
j_3	0	9	20	5	1	0	0	-	-
j_4	0	18	20	10	1	0	-	-	-

3.3. A Micro-opportunistic Approach

In the operation-centered approach implemented in MICRO-BOSS, each operation is considered an independent decision point. Any operation can be scheduled at any time, if deemed appropriate by the scheduler. There is no obligation to simultaneously schedule other operations upstream or downstream within the same job, nor is there any obligation to schedule other operations competing for the same resource.

MICRO-BOSS proceeds by iteratively selecting an operation to be scheduled and a reservation (i.e. start time) to be assigned to that operation. Every time an operation is scheduled, a new *search state* is created, where new constraints are added to account for the reservation assigned to that operation. A so-called consistency labeling procedure is applied to that state, that updates the set of remaining possible reservations of each unscheduled operation. If an unscheduled operation is found to have no possible reservations left, a *deadend state* has been reached: the system needs to *backtrack* (i.e. it needs to undo some earlier reservation assignments in order to be able to complete the schedule). If the search state does not appear to be a deadend, the scheduler moves on and looks for a new operation to schedule and a reservation to assign to that operation. This process goes on until all operations have been scheduled, or until the scheduling problem has been found to be infeasible.

Because job shop scheduling is NP-complete, this procedure could require exponential time in the *worst-case*. In practice, as demonstrated by the experimental study presented in this chapter[10], it is possible to maintain the *average complexity* of the procedure at a very low

[10]See also [Sadeh & Fox 90a] for a more detailed study of this issue.

level while producing quality schedules. This is achieved by interleaving search with the application of consistency labeling techniques and a set of look-ahead techniques that help decide which operation to schedule next (so-called *variable ordering heuristic* [Dechter & Pearl 87]) and which reservation to assign to that operation (so-called value ordering heuristic).

1. **Consistency Labeling (or Consistency Checking):** Consistency labeling techniques are used to prune reservations that have become unavailable due to earlier reservation assignments. By constantly propagating new constraints resulting from earlier scheduling decisions, these techniques reduce the chance of selecting reservation assignments that are inconsistent with earlier scheduling decisions. This in turn diminishes the chances of backtracking. Additionally, by allowing for the early detection of deadend states, these techniques limit the amount of work wasted in the exploration of fruitless alternatives. In other words these techniques reduce both the frequency and the amount of backtracking.

2. **Look-ahead Analysis**: A two-step look-ahead procedure is applied to each search state, which first optimizes reservation assignments within each job, and then, for each resource, computes contention between jobs over time. Resource/time intervals where job contention is the highest help identify the critical operation to be scheduled next (variable or operation ordering heuristic). Reservations for that operation are then ranked according to their ability to minimize the costs incurred by the conflicting jobs (value or reservation ordering heuristic). As the schedule is constructed, contention between jobs for some resource/time intervals drops while contention for others may increase. By constantly redirecting its effort towards the most serious conflicts, the scheduler attempts to always focus on the most important tradeoffs. Eventually contention subsides, and the problem becomes more decoupled. This typically allows for the construction of schedules that are closer to a global optimum. Simultaneously, because the scheduling strategy is aimed at reducing job contention as fast as possible, chances of backtracking tend to subside pretty fast too.

In MICRO-BOSS, the so-called opportunistic behavior results from the ability of the scheduler to constantly revise its search strategy and redirect its effort towards the scheduling of the operation that appears to be the most critical in the current search state. This degree of opportunism differs from that displayed by other approaches where the scheduling entity is an entire resource or an entire job [Ow & Smith 88, Collinot et al. 88], i.e. where an entire resource or an entire job needs to be scheduled before the sc' eduler is allowed to revise its current scheduling strategy.

Concretely, MICRO-BOSS starts in a search state in which no operation has been scheduled yet, and proceeds according to the following steps:

1. If all operations have been scheduled then stop, else go on to 2;
2. Apply the **consistency labeling** procedure;
3. If a deadend is detected then **backtrack** (i.e. select an alternative if there is one left and go back to 1, else stop and report that the problem is infeasible), else go on to step 4;
4. Perform the **look-ahead** analysis: rank the possible reservations of each unscheduled operation according to how well they minimize the costs of the job to which the operation belongs, and evaluate job contention over time for each resource;
5. Select the next operation to be scheduled (so-called **operation ordering** heuristic);
6. Select a reservation for that operation (so-called **reservation ordering** heuristic)
7. Create a **new search state** by adding the new reservation assignment to the current partial schedule. Go back to 1.

The results reported in this chapter were obtained using a simple chronological backtracking scheme. The remainder of this section gives a more detailed on the look-ahead analysis step and the operation and reservation ordering heuristics. Further details on these techniques as well as other aspects of MICRO-BOSS can be found in [Sadeh 91].

3.3.1. Look-ahead Analysis in MICRO-BOSS

A two-step look-ahead procedure is applied to each search state, which first optimizes reservation assignments within each job, and then, for each resource, computes contention between jobs over time. The so-called *demand profiles* produced by these computations help identify operations whose good reservations (within their jobs) conflict with the good reservations of other operations. Scheduling these operations requires trading the costs incurred by the competing jobs. Because these tradeoffs critically affect the quality of the entire schedule, they should be worked out as early as possible. This way, the scheduler ensures that it still has many options left to select judicious tradeoffs. The remaining operations also tend to become more decoupled and hence easier to optimize. By constantly redirecting search towards the most serious tradeoff operations, the scheduler produces better schedules and simultaneously reduces its chances of backtracking.

This subsection describes the two-step look-ahead analysis implemented in MICRO-BOSS.

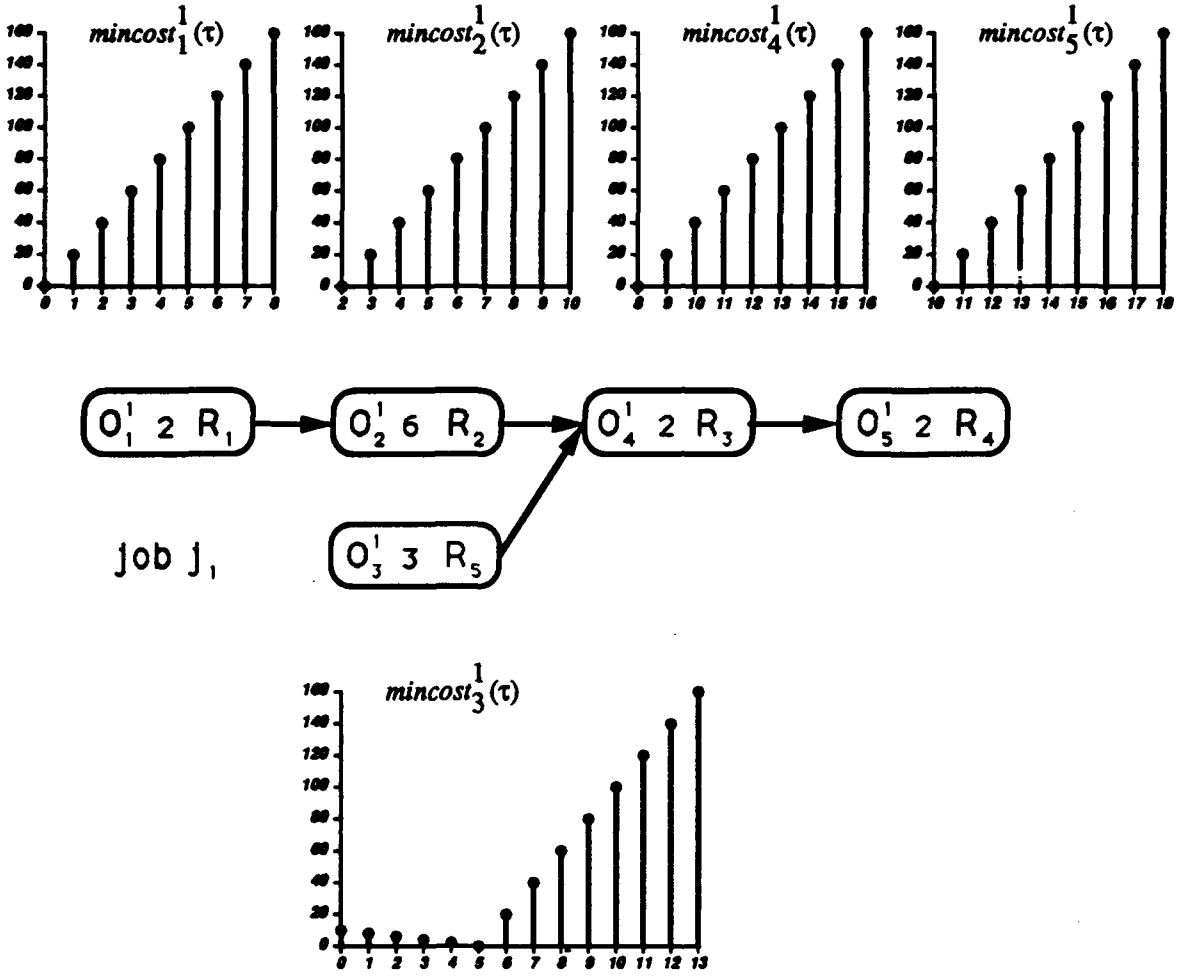
Following subsections describe the operation and reservation ordering heuristics based on this look-ahead analysis.

3.3.1.1. Step 1: Reservation Optimization Within a Job

In addition to keeping track of the start times that remain available to an unscheduled operation, say O_i^k, the scheduler implicitly maintains a cost, $mincost_i^k(\tau)$, for each of these possible start times. This cost, $mincost_i^k(\tau)$, is the minimum cost that would be incurred by the job, if O_i^k was to start at $st_i^k = \tau$ rather than at one of the best start times currently available to that operation. As other operations both within the job and in other jobs are scheduled, these costs change. Some start times become unavailable. The relative merits of start times that remain possible may change due to scheduling decisions made within the same job or in other jobs. As the scheduler moves from one search state to the other, it is critical to reassess the merits of the remaining possible start times of each unscheduled operation in order to identify where important tradeoffs still need to be made. This requires identifying whether the choice of a start time for a given unscheduled operation still affects the tardiness costs of the job to which that operation belongs. It also requires determining which inventory costs, if any, are still affected by that choice. Once these costs have been determined, the scheduler can identify which start time(s) among the ones that are still possible will be optimal for the operation (within the job), and the minimum costs that would be incurred by the job if a suboptimal start time was selected.

When dealing with in-tree process routings, the start time assigned to an operation will not affect the tardiness costs incurred by the job (to which that operation belongs) if another operation downstream within that job (i.e. a successor of that operation) has already been scheduled. On the other hand, when operations upstream within the job have already been scheduled, the inventory costs introduced by these operations or by operations further upstream within the job may still be affected by the start time selected for a downstream operation, if that start time assignment forces the job to complete past its due date (see the $Max(C_l, dd_l)$ term in equation ((3.2))). This is best explained with an example.

Figure 3-3 displays the $mincost_i^1$ curves for each of the five operations required by job j_1 in the example introduced in section 2. These curves were computed in the initial search state. In that state, all operations still have to be scheduled, and all resources are entirely available. Consider operation O_1^1. After consistency labeling, the earliest possible start time of that operation is $est_1^1 = 0$ and its latest possible start time $lst_1^1 = 8$: O_1^1 has 9 possible start times. In order for job j_1 to meet its due date (at time 12), operation O_1^1 would have to be started at $st_1^1 = 0$, and the other operations in job j_1 would have to be compactly scheduled right after it. This optimal schedule is a just-in-time schedule. Any delay in starting O_1^1 will



$erd_1 = 0$	$dd_1 = 12$	$lcd_1 = 20$	$tard_1 = 20$	
$inv_1^1 = 2$	$inv_2^1 = 1$	$inv_3^1 = 2$	$inv_4^1 = 0$	$inv_5^1 = 0$

Figure 3-3: Assessing the merits of alternative scheduling decisions in the initial search state.

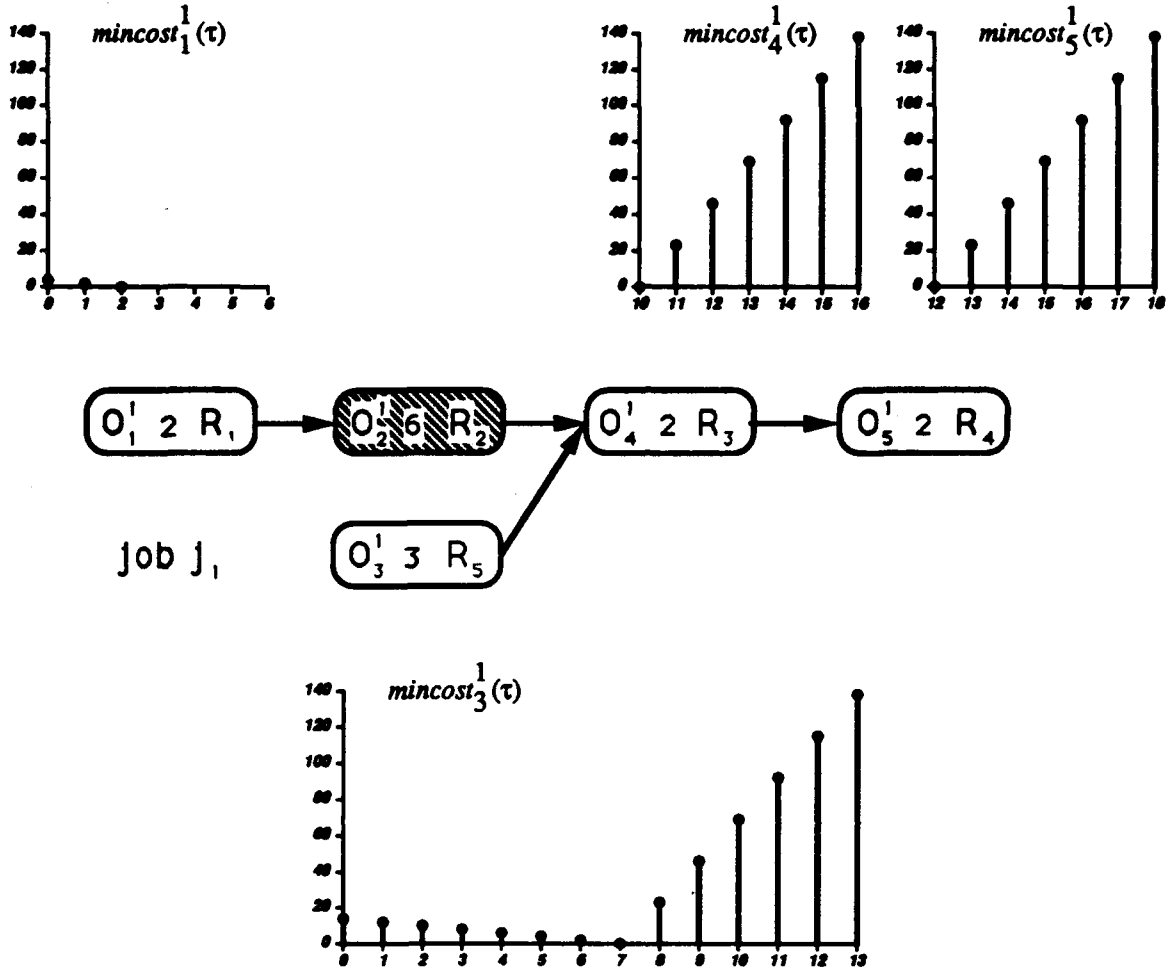
create tardiness costs. For instance, starting O_1^1 at $st_1^1 = 1$ would result in tardiness costs of at least $20 \times 1 = 20$ (since, in the best case, job j_1 would be late by 1 time unit and $tard_1 = 20$). Ideally job j_1 would still be compactly scheduled, and inventory costs would not be larger than those in the just-in-time schedule. Hence, in this initial state, the minimum additional

costs incurred by job j_1 if $st_1^1 = 1$ is $mincost_1^1(1) = 20$. Similarly $mincost_1^1(2) = 40$, $mincost_1^1(2) = 60$, etc. Similar considerations apply to operations O_2^1, O_4^1, and O_5^1. For operation O_3^1, things are slightly different. Indeed, if that operation is started before $st_3^1 = 5$ (and hence completes before time 8), tardiness costs no longer decrease. Instead, additional inventory costs will be incurred by job j_1. For instance, if $st_3^1 = 4$, the best compatible schedule for j_1 will complete on time but will incur an overhead in inventory cost of $2 \times 1 = 2$ (since $inv_3^1 = 2$), hence $mincost_3^1(4) = 2$. Similarly $mincost_3^1(3) = 4$, $mincost_3^1(2) = 6$, etc.

Suppose that the scheduler creates a new search state and schedules[11] O_2^1 to start at $st_2^1 = 4$ (see Figure 3-4). In this search state (and all its possible descendants, i.e. all the possible schedules that can be built by completing this partial schedule), job j_1 can no longer be completed on time. Job j_1 will be at least 2 time units behind its due date. This affects the optimal start time of O_3^1 which shifts from $st_3^1 = 5$ to $st_3^1 = 7$. Starting O_3^1 earlier than at $st_3^1 = 7$ no longer reduces tardiness costs. Instead it would only increase inventory costs (e.g. $mincost_3^1(6) = 2$ and $mincost_3^1(0) = 14$). Notice that the penalty incurred by job j_1 for every time unit that O_3^1 starts after its best start time (i.e. $st_3^1 = 7$ in the current state, $st_3^1 = 5$ in the initial state) has increased compared to what it used to be in the initial state. Indeed, in the current search state, if O_3^1 is scheduled to start past its best start time, say at $st_3^1 = 8$ for instance (i.e. 1 time unit past its best start time), job j_1 will not only incur a tardiness cost of $1 \times tard_1 = 20$, but it will also see its inventory costs augment by at least $(inv_1^1 + inv_2^1) \times 1 = 3$. This is because the start time of O_2^1 is now fixed. If O_3^1 were to be started at $st_3^1 = 8$, O_1^1 and O_2^1 could no longer be compactly scheduled right in front of O_4^1 : in the best case there would be a gap of 1 time unit between O_2^1 and O_4^1. In other words, the apparent marginal tardiness cost at operation O_3^1 has changed from 20 to 23 ! For the same reason, the apparent marginal tardiness costs at O_4^1 and O_5^1 are 23 also. The best remaining possible start times for these two operations are respectively 10 and 12. The minimum additional costs for scheduling O_5^1 at, say $st_5^1 = 18$ (instead of its best remaining start time $st_5^1 = 12$) is $mincost_5^1(18) = 6 \times 23 = 138$ (while it was 160 in the initial search state).

The *mincost* curve for O_1^1 has changed more dramatically. Whichever start time is assigned to O_1^1, this start time no longer affects the tardiness costs incurred by the job. While in the initial search state, scheduling O_1^1 early was reducing the minimum tardiness costs incurred by the job, now it only increases the minimum inventory costs of that job (e.g. $mincost_1^1(0) = 4$, since $inv_1^1 = 2$).

[11]This scheduling decision is not a good one. It is only made to illustrate interesting changes in the *mincost* functions.



$erd_1 = 0$	$dd_1 = 12$	$lcd_1 = 20$	$tard_1 = 20$	
$inv_1^1 = 2$	$inv_2^1 = 1$	$inv_3^1 = 2$	$inv_4^1 = 0$	$inv_5^1 = 0$

Figure 3-4: Assessing the merits of alternative scheduling decisions in a search state where O_2^1 has been scheduled to start at $st_2^1 = 4$.

As the schedule is constructed, the merits of different possible start times can dramatically change. It is critical for the scheduler to constantly keep track of these changes in order to redirect search towards the most critical operations, and select reservations for these

operations that provide a good compromise between the cost incurred by the critical operation and the other operations with which it competes.

For sake of efficiency, the current implementation of MICRO-BOSS updates the *mincost* functions under the simplifying assumption that the set of possible start times of each unscheduled operation is made of a single contiguous time interval (see [Sadeh 91] for details). Under this simplifying assumption, it is not necessary to explicitly compute the *mincost* function at each possible start time of each unscheduled operation. Instead, the scheduler maintains for each unscheduled operation O_i^k, an apparent marginal tardiness cost, $ap\text{–}tard_i^k$, an apparent marginal inventory cost, $ap\text{–}inv_i^k$, and keeps track of the best possible start time(s) that are currently available to that operation. The apparent marginal tardiness cost, $ap\text{–}tard_i^k$ (apparent marginal inventory cost, $ap\text{–}inv_i^k$), of operation O_i^k, in a given search state, is the minimum cost increase incurred by job j_k for every unit of time that O_i^k is scheduled past (beyond) its best possible start time(s), under the simplifying assumption of single contiguous intervals of possible start times.

When moving from one search state to the other, apparent marginal inventory costs and apparent marginal tardiness costs only change in the job in which an operation was just scheduled. The best remaining start times of unscheduled operations may however change both within that job and in other jobs. Updating the apparent costs in the job in which an operation was just scheduled, can be done in $O(n_j)$ time, where n_j is the number of operations in that job. Altogether updating the apparent marginal costs within the job where an operation was just scheduled, and the best remaining start times of all unscheduled operations, takes at most $O(n)$ time, where n is the number of unscheduled operations.

3.3.1.2. Step 2: Building Demand Profiles to Identify Highly Contended Resource/Time Intervals

If all operations could be simultaneously scheduled at one of their best start times, there would be no scheduling problem. Generally this is not the case. Jobs typically have conflicting requirements. The micro-opportunistic approach attempts to identify critical operations whose good start times (within their jobs) conflict with the good start times of other operations, and focuses on optimizing these conflicts first. The importance of a conflict (and the criticality of the operations participating in that conflict) depends on the number of jobs that are competing for a same resource over some time interval, the amount of temporal overlap between the requirements of these jobs, the number of alternative reservations (i.e. start times) still available to the conflicting operations and the extra costs incurred by the jobs if they have to chose these alternative reservations (as determined the *mincost* functions computed in the previous step).

A simple way to identify resource/time intervals that are highly contended for would be to

assume that all operations will be simultaneously scheduled at their best start times. Resource/time intervals that would need to be reserved more than once would indicate the need for a tradeoff. In search states, where some operations have more than one best start time, this approach would not work. Because we know that, in general, it is impossible to schedule all operations at one of their best start times, we may also fail to identify conflicts that will arise later on as some operations are scheduled at start times that appear to be less than optimal in the current search state. Last but not least, demand profiles obtained by simultaneously scheduling each operation at one of its best start time would not provide any information on the marginal costs of the operations involved in a particular conflict. A conflict involving a small number of operations, all with high marginal costs, might turn out to be more critical than a conflict with a larger number of operations whose marginal costs are lower. For these reasons, we have adopted a probabilistic framework, in which each start time τ that remains possible for an unscheduled operation O_i^l is assigned a *subjective probability* $\sigma_i^l(\tau)$ to be selected for that operation. Possible start times with lower *mincost* values are simply assigned a larger probability, thereby reflecting our expectation that they will allow for the production of better schedules. Using these start time distributions, the scheduler builds for each unscheduled operation O_i^l, an *individual demand profile* $D_i^l(t)$, that indicates the subjective probability that the operation will be requiring its resource in function of time. By aggregating the individual demand profiles of all unscheduled operations requiring a given resource, R_k, the scheduler obtains an *aggregate demand profile*, $D_{R_k}^{aggr}(t)$, that indicates contention between unscheduled operations for that resource in function of time. More formally, if we assume a stochastic mechanism that completes the current partial schedule by randomly assigning a start time to each unscheduled operation O_i^l, according to its σ_i^l distribution, $D_{R_k}^{aggr}(t)$ is the expected number of reservations that would be made for resource R_k at time t.

In the current implementation, the start time probability distribution of a typical unscheduled operation O_i^l is of the form:

$$\sigma_i^l(\tau) = N_i^l \times [1 - B \times \frac{mincost_i^l(\tau)}{Maxcost}] \qquad (3.3)$$

where:

- N_i^l is a normalization factor that ensures that $\sum_{\tau} \sigma_i^l(\tau) = 1$, since O_i^l will be assigned exactly one start time ;

- *Maxcost* is the maximum value of the *mincost* functions taken over all the remaining possible start times of all remaining unscheduled operations;

- B is a system parameter that controls the degree to which start time distributions are biased towards good start times ($0 \leq B < 1$). In particular if $B = 0$, all possible start times are considered equally probable (uniform start time distributions). We set B to 0.9 both in the example presented below and in the experiments reported later in this chapter.

Because the micro-opportunistic scheduler optimizes the most important conflicts first, the costs that remain to be optimized tend to decrease during the construction of the schedule. By updating the value of *Maxcost* in each search state and using it to measure cost differences in equation ((3.3)), the scheduler constantly adjusts its sensitivity to the largest differences in costs that are still possible in the current state[12].

Given these start time probability distributions, the probability that an unscheduled operation O_i^l uses its resource at time t, which is referred to as the *individual demand* of O_i^l for R_i^l, is:

$$D_i^l(t) = \sum_{t - du_i^l < \tau \leq t} \sigma_i^l(\tau) \tag{3.4}$$

Notice that the normalization factor N_i^l ensures that the total demand of each unscheduled operation O_i^l is equal to the duration of that operation, du_i^l (i.e. $\sum_\tau D_i^l(\tau) = du_i^l$). This total demand has simply been spread over the different start times that remain available to that operation in the current search state, according to the distribution defined in equation ((3.3)). Operations with a lot of good start times (i.e. operations that have a lot of start times with a low *mincost* value, compared to the value of *Maxcost* in the current search state) will have individual demand profiles that are almost uniformly spread over long time intervals. Their individual demands will be fairly low at any given moment in time. In other words, because these operations have a large number of good start times still available, they do not rely heavily on anyone of them. Instead operations with a small number of good start times will have more compact individual demands. Their demand will be high over relatively short intervals, thereby reflecting a higher reliance on the availability of these intervals.

---

[12]When dealing with jobs that have a lot of slack (e.g. a job whose latest acceptable completion date is far beyond both its due date and its earliest possible completion date), it is important to only include meaningful cost differences in the computation of *Maxcost*. To this end, the current implementation, uses a slightly modified definition of *Maxcost*, which is computed as the maximum value of the *mincost* functions for start times that allow their job to start after $dd_l - 5\sum_{i=1}^{n_l} du_i^l$ and complete by $dd_l + \sum_{i=1}^{n_l} du_i^l$. The probability assigned to earlier (later) start times is obtained by interpolating the start time distribution so that $est_i^l - 1$ ($lst_i^l + 1$) has a probability of zero.

For each resource R_k, the total demand of all unscheduled operations for that resource is given by:

$$D_{R_k}^{aggr}(t) = \sum_{R_i^l = R_k} D_i^l(t) \tag{3.5}$$

where the summation is carried over all unscheduled operations that need resource R_k.

Observe also that the consistency labeling technique used by the scheduler ensures that all start times that would conflict with reservations that have already been made in the current partial schedule have been pruned from the set of remaining possible start times of all unscheduled operations (see subsection 3.1). For this reason, the aggregate demands never overlap with reservations that have already been made.

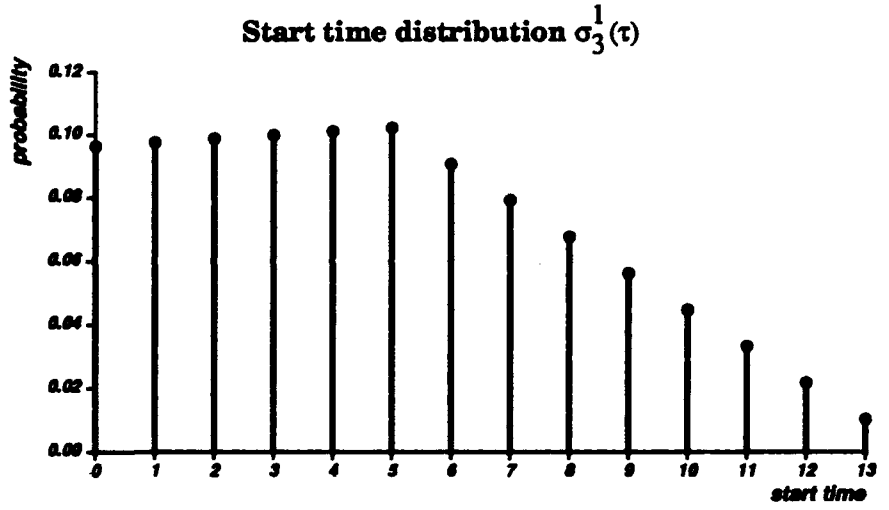


Figure 3-5: Start time distribution $\sigma_3^1(\tau)$ for operation O_3^1 in the initial search state depicted in Figure 3-3.

Figure 3-5 represents $\sigma_3^1(\tau)$, the start time distribution of operation O_3^1, in the initial search state depicted in Figure 3-3. From the *mincost* curves represented in Figure 3-3 and those of the operations in the three other jobs, it can be seen that the largest value taken by a *mincost* function in the initial state is 160, i.e. $Maxcost = 160$. In other words, the largest difference in cost between the worst and the best possible start time of any operation is never larger than 160 in the initial state. The best start time of O_3^1 is $st_3^1 = 5$, i.e. $mincost_3^1(5) = 0$. Hence, using equation ((3.3)), we obtain $\sigma_3^1(5) = N_3^1$. The next best start time for that operation is $st_3^1 = 4$ with $mincost_3^1(4) = 2$, hence $\sigma_3^1(4) = N_3^1 \times (1 - 0.9 \times \frac{4}{160}) = N_3^1 \times 0.94375$. By performing similar

computations for all the other possible start times of O_3^1, we find that $N_3^1 = \frac{1}{9.78125}$. Hence $\sigma_3^1(4) = 0.1011$, $\sigma_3^1(5) = 0.1022$, $\sigma_3^1(6) = 0.0907$, etc. Similar computations can be performed for all the other unscheduled operations.

The resulting distributions are used to build the individual demand profiles of each unscheduled operation according to equation ((3.4)). These demand profiles are in turn aggregated according to equation ((3.5)). This process is illustrated in Figure 3-6 for resource R_2. There are four operations requiring resource R_2: O_2^1, O_2^2 O_3^3, and O_2^4. Figure 3-6 displays the individual demand profile of each of these four operations, and the aggregate demand for R_2 obtained by summing the four individual demands. As expected, the individual demands of operations O_3^3 and O_2^4 are quite uniform since these two operations have relatively low apparent marginal costs. In contrast, the individual demands of operations O_2^1 and O_2^2, which have larger apparent marginal costs, are more concentrated around the good reservations of these operations.

The aggregate demand for each of the five resources are shown in Figure 3-7. As expected, resource R_2 appears to be the most contended for. The aggregate demand for that resource is well above 1.0 over a large time interval, with a peak at 1.49. Resource R_1 appears to be a potential bottleneck at the beginning of the problem, with a demand peaking at 1.20. Whether R_1 will actually be an auxiliary bottleneck or not cannot be directly determined from the curves displayed in Figure 3-7. Instead the scheduler needs to update these curves in each search state to account for earlier decisions. It could be the case that, as operations requiring R_2 are scheduled, the aggregate demand for R_1 becomes smoother. We will see that this does not happen in this example. On the contrary, after only a portion of the operations requiring resource R_2 have been scheduled, the scheduler will shift its attention to resource R_1.

Updating the start time distributions, the individual and aggregate demand profiles, requires at most $O(nk)$ time in each search state, where n is the number of unscheduled operations and k an upper-bound on the number of possible start times left to each unscheduled operation.

The idea of building biased demand profiles is not new. [Muscettola & Smith 87] describes a technique in which a random schedule generator is used to detect potential resource congestion, given a predefined scheduling strategy. The random generator is biased to select more often those reservations that are expected to produce better schedules. Our technique is different, as it does not assume a predefined scheduling strategy, but instead uses the resulting demand profiles to dynamically revise the current strategy.

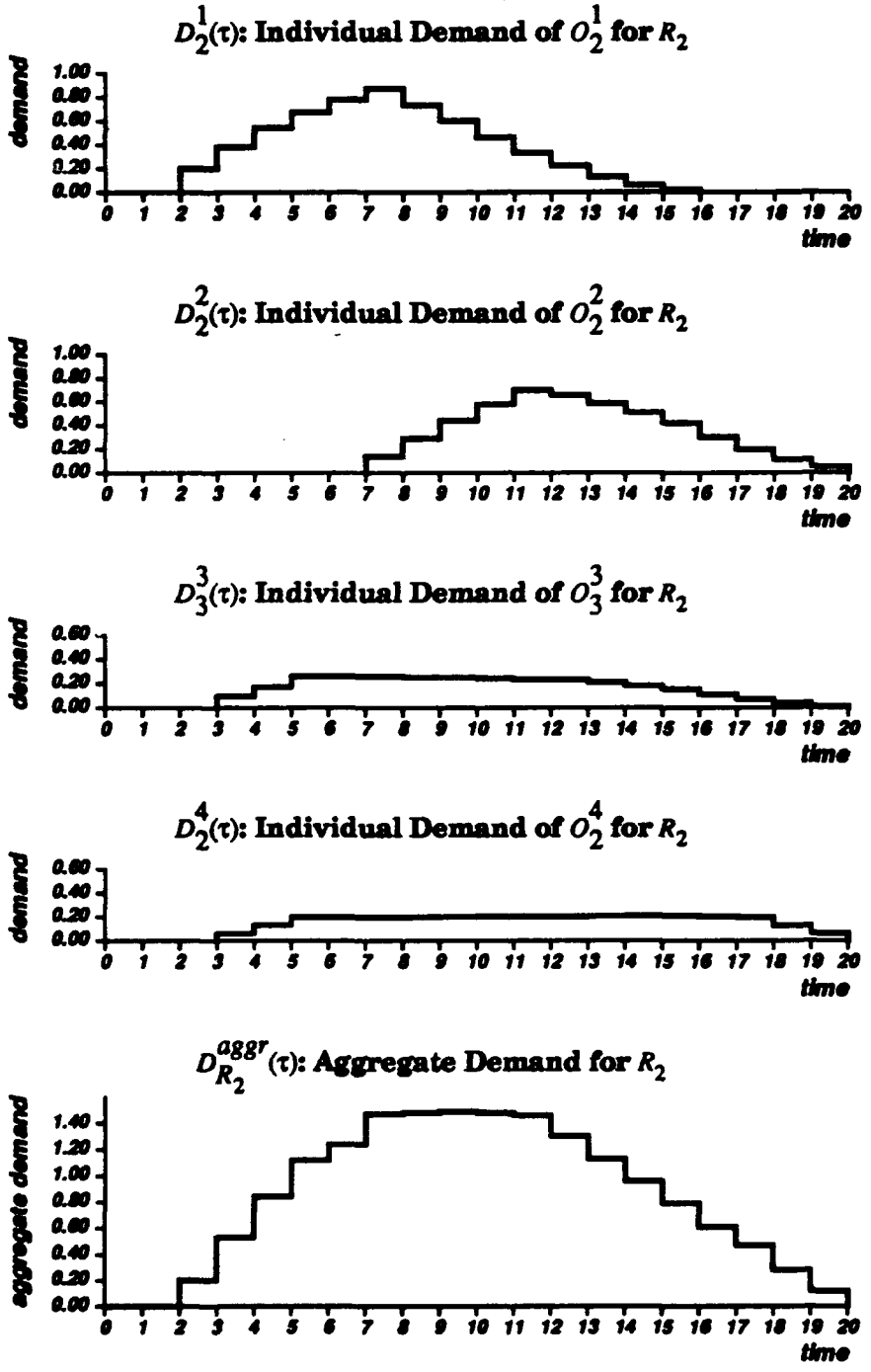


Figure 3-6: Building R_2's aggregate demand profile in the initial search state.

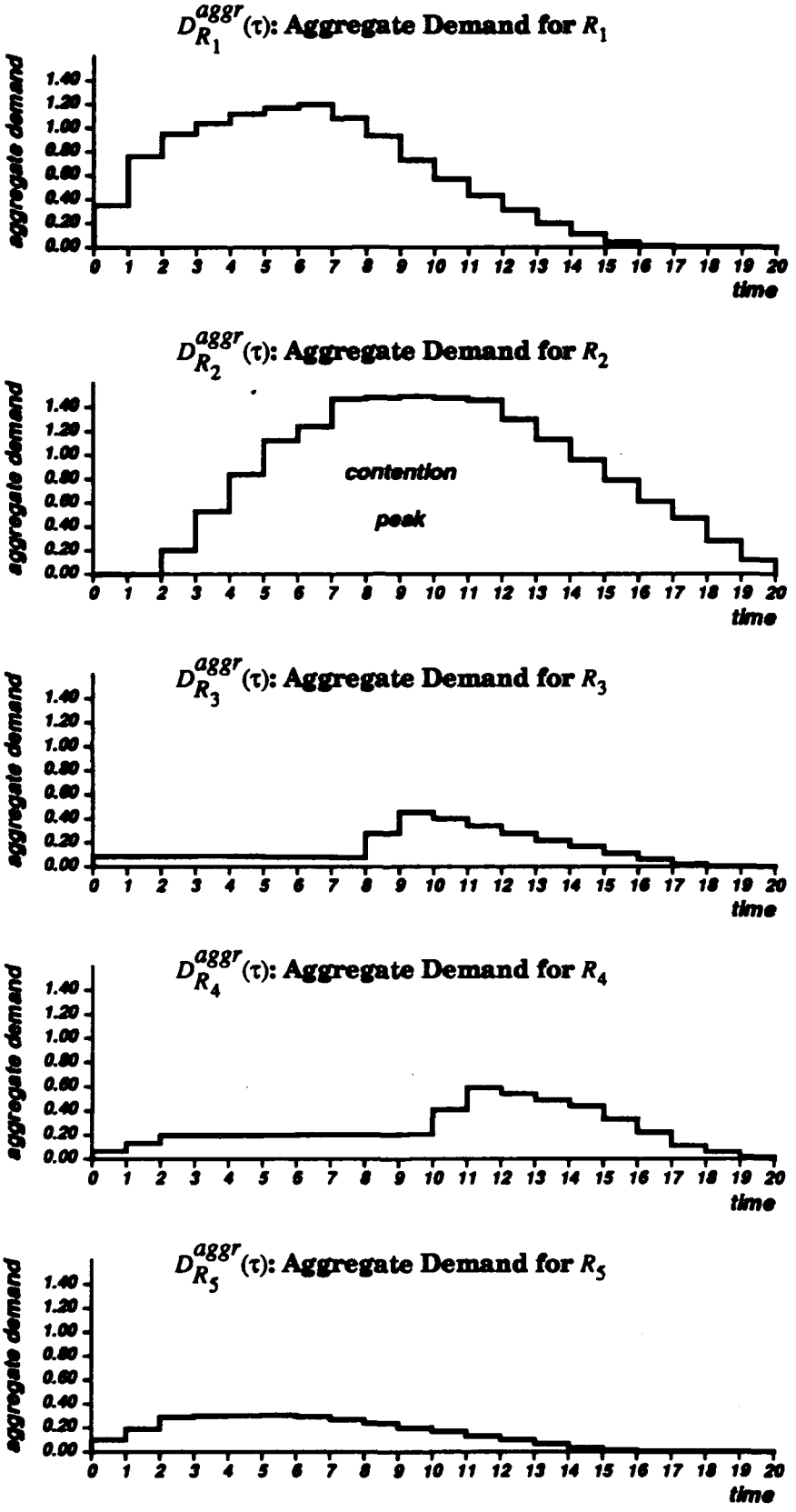


Figure 3-7: Aggregate demands in the initial search state for each of the five resources.

3.3.2. Operation Selection

Critical operations are identified as operations whose good reservations conflict with the good reservations of other operations. The largest peak in the aggregate demand profiles determines the next conflict (or bottleneck) to be optimized, and the operation with the largest reliance on the availability of that peak is selected to be scheduled next. Intuitively the operation that relies most on the availability of the most contended resource/time interval is also the one whose good start times are the most likely to become unavailable if other operations contending for that resource/time interval were scheduled first.

More specifically, the scheduler decomposes the demand profile of each resource into time intervals of length equal to the average duration of the operations requiring that resource. The time interval with the largest *average* aggregate demand is identified as the one for which contention is the highest, and the operation with the largest individual demand for that resource/time interval is selected to be scheduled next[13]

In the example introduced earlier, the most contended demand peak is that for resource R_2 over interval [7,12]. Figure 3-8 displays the aggregate demand for resource R_2 together with the individual demands of the four operations contributing to this demand. The operation with the largest contribution to the demand peak is O_2^1. Therefore this operation is selected to be scheduled next. This is no real surprise: O_2^1 belongs to one of the two jobs in the problem that have a high marginal tardiness cost ($tard_1 = 20$). While any delay in starting job j_1 will make this job tardy, job j_2 (i.e. the other job with a high marginal tardiness cost) can tolerate a small amount of delay without ending up late.

3.3.3. Reservation Selection

Once it has selected an operation, the scheduler attempts to identify a reservation for that operation that will minimize as much as possible the costs incurred by the job to which that operation belongs and by other competing jobs. This is equivalent to solving a one-machine early/tardy problem in which operations scheduled past their best start times incur penalties determined by their apparent marginal tardiness costs, while operations scheduled before their best start times incur earliness penalties determined by their apparent marginal inventory costs.

The one-machine early/tardy problem is an NP-complete problem in itself [Garey et al. 88]. Because of the presence of earliness costs, solving this problem to optimality requires, in

[13]Multiple variations of of this heuristic have been tried, including variations using different interval lengths and different methods for selecting the critical operation, once the demand peak has been identified. The variation described here appears to be best both with respect to schedule quality and search efficiency.

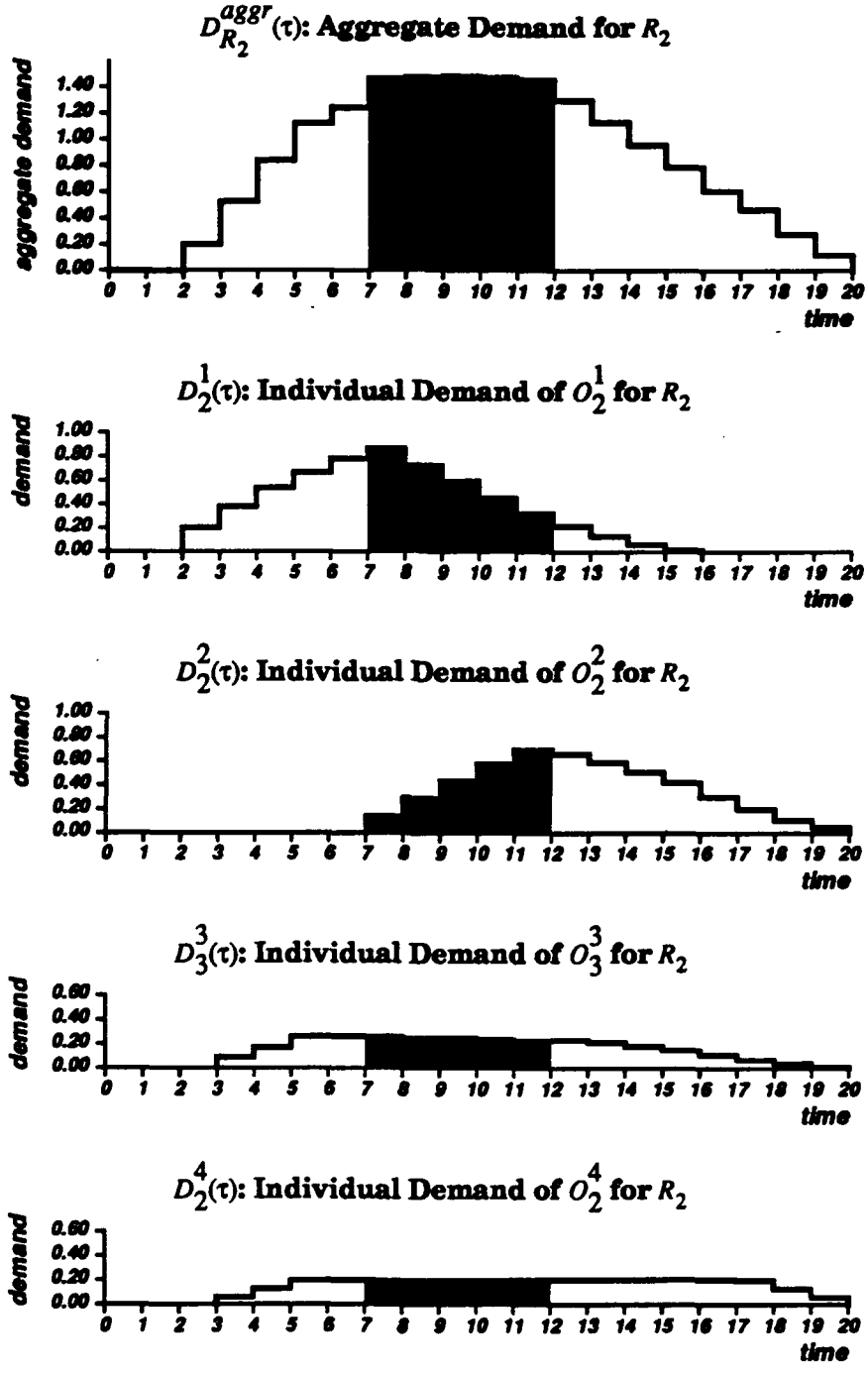


Figure 3-8: Operation selection in the initial search state.

general, the insertion of idle-time in the schedule. It can be shown that , given a fixed

operation sequence, idle time can be optimally inserted in that sequence in $O(N \log N)$ time, where N is the number of operations requiring the resource [Garey et al. 88]. Hence the problem is reduced to that of finding a sequence that will minimize the early/tardy costs, once idle time has been optimally inserted. Most of the procedures proposed so far to deal with this problem rely on branch-and-bound enumeration procedures [Baker & Scudder 90]. [Fry et al. 87] describes a dominance criterion that helps speed up this approach. Nevertheless the computational requirements of this procedure remain quite prohibitive even for small problems [Fry et al. 87][14]. On the other hand, Ow and Morton have developed a family of much faster procedures that are pretty good at minimizing early/tardy costs under the simplifying assumption that no idle time will be inserted [Ow & Morton 89].

For these reasons, a hybrid reservation ordering heuristic was implemented that adapts to the amount of contention for the critical resource/time interval. When contention is particularly high, a variation of the early/tardy procedure described in [Ow & Morton 89] is used to identify a good reservation for the operation to be scheduled. Indeed, when contention is high, inserting idle time in the one-machine schedule is not likely to significantly improve that schedule. When contention is lower, the scheduler dynamically switches to a greedy reservation ordering heuristic, in which reservations are simply rated according to their apparent costs (i.e. according to their *mincost* values). Indeed, in situations where contention is not too high, a sizable proportion of the good start times of non critical operations tend to still be available after more critical operations have been scheduled. When this is the case, a greedy reservation ordering tends to produce high quality solutions. In particular, it inserts idle time as required by the operation it is scheduling[15].

The worst-case complexity of this reservation ordering heuristic is $O(max\{N^2, k \log k\})$, where N is the number of jobs competing for the critical resource and k the number of start times that remain possible for the critical operation[16].

3.3.4. A Small Example

The MICRO-BOSS scheduling system described in this chapter has been implemented in Knowledge Craft, a frame-based language that runs on top of Common Lisp. The system runs on a DECstation 3100 under Mach UNIX. The small example used throughout this chapter requires a little over 2 seconds of CPU time in the current implementation. An edited trace of that example appears in Figure 3-9.

---

[14]Close to 10 minutes of CPU time on a VAX 780 for a 15-operation problem.

[15]See [Sadeh 91] for further details.

[16]$O(N^2)$ corresponds to the complexity of the early/tardy procedure, and $O(k \log k)$ is the time required to sort the possible start times. When the scheduler switches to its greedy reservation ordering heuristic, the complexity of the reservation ordering heuristic becomes simply $O(k \log k)$.

Observe that, rather than entirely scheduling the main bottleneck resource, namely resource R_2, the scheduler shifted to resource R_1 only after two out of the four operations requiring R_2 had been scheduled. The average expected demand displayed in each search state is the average demand for the critical demand peak, and the average contribution is that of the critical operation for the demand over that peak. The decoupling effect of the operation ordering heuristic is very clear in this example. In particular, the average demand over the critical peak consistently decreases from one search state to the next, thereby indicating a regular decrease in contention as the schedule is constructed (remember that the demand peak corresponds to the interval of highest contention in the current search state). This observation is correlated by the average contribution of the critical operation to the demand for the peak in each search state. As the schedule is constructed, the contribution of the critical operation to the peak becomes a larger proportion of the total demand for that peak. This indicates that there are fewer and fewer operations contending with each other. After half of the operations have been scheduled (depth 7), contention has totally disappeared: the critical operation is the only one to contribute to the demand for the peak. In other words the problem has been totally decoupled. The resource requirements of the operations that still need to be scheduled no longer interact with each other. This phenomenon is not specific to this example, but can be observed in all the problems that we have run. This suggests that the operation ordering heuristic is indeed very good at constantly redirecting search towards the most important conflicts.

Notice also that no backtracking was necessary to produce the schedule. The schedule produced by the micro-opportunistic approach is displayed in Figure 3-10.

The cost of this schedule is 180. In comparison the Earliest Due Date priority rule produces a schedule with a cost of 208, the Weighted Shortest Processing Time rule and the Weighted COVERT priority rule [Vepsalainen & Morton 87]produce a schedule with a cost of 255, and the Slack per Remaining Processing Time rule produces a schedule with a cost of 300. In this example the savings allowed by the micro-opportunistic approach are the results of reductions both in tardiness and inventory costs. Extensive experimental results are presented in the next section that study the micro-opportunistic approach under a variety of scheduling conditions.

3.4. Performance Evaluation

This section reports two experimental studies that were carried out in order to evaluate the performance of MICRO-BOSS Subsection 4.1 describes how scheduling problems were randomly generated to cover a wide variety of scheduling conditions. Subsection 4.2 describes a study comparing MICRO-BOSS against four priority dispatch rules that are known to be particularly good at reducing tardiness. Subsection 4.3 attempts to answer the

MON NOV 12 1990 --- 17:04:49 EST

>> Depth: 0, Number of states visited: 0
Critical demand peak:
R_2 between 7 and 12, Avg. expected demand: 1.48
Critical Operation: O_2^1, Avg. contrib.: 0.60
Using early/tardy reservation ordering heuristic:
O_2^1 scheduled between 2 and 8 on R_2

>> Depth: 1, Number of states visited: 1
Critical demand peak:
R_2 between 10 and 15, Avg. expected demand: 1.33
Critical Operation: O_2^2, Avg. contrib.: 0.64
Using early/tardy reservation ordering heuristic:
O_2^2 scheduled between 9 and 14 on R_2

>> Depth: 2, Number of states visited: 2
Critical demand peak:
R_1 between 0 and 4, Avg. expected demand: 1.35
Critical Operation: O_1^2, Avg. contrib.: 0.75
Using early/tardy reservation ordering heuristic:
O_1^2 scheduled between 2 and 9 on R_1

>> Depth: 3, Number of states visited: 3
Critical demand peak:
R_2 between 14 and 19, Avg. expected demand: 1.13
Critical Operation: O_3^3, Avg. contrib.: 0.58
Using early/tardy reservation ordering heuristic:
O_3^3 scheduled between 17 and 20 on R_2

>> Depth: 4, Number of states visited: 4
Critical demand peak:
R_2 between 14 and 19, Avg. expected demand: 0.60
Critical Operation: O_2^4, Avg. contrib.: 0.60
Using greedy reservation ordering heuristic:
O_2^4 scheduled between 14 and 17 on R_2

>> Depth: 5, Number of states visited: 5
Critical demand peak:
R_4 between 10 and 13, Avg. expected demand: 0.57
Critical Operation: O_5^1, Avg. contrib.: 0.34
Using greedy reservation ordering heuristic:
O_5^1 scheduled between 10 and 12 on R_4

Figure 3-9: An edited trace

>> Depth: 6, Number of states visited: 6
Critical demand peak:
R_3 between 8 and 10, Avg. expected demand: 1.08
Critical Operation: O_4^1, Avg. contrib.: 1.0
Using greedy reservation ordering heuristic:
O_4^1 scheduled between 8 and 10 on R_3

>> Depth: 7, Number of states visited: 7
Critical demand peak:
R_5 between 4 and 7, Avg. expected demand: 0.55
Critical Operation: O_3^1, Avg. contrib.: 0.55
Using greedy reservation ordering heuristic:
O_3^1 scheduled between 5 and 8 on R_5

>> Depth: 8, Number of states visited: 8
Critical demand peak:
R_1 between 0 and 4, Avg. expected demand: 0.50
Critical Operation: O_1^1, Avg. contrib.: 0.50
Using greedy reservation ordering heuristic:
O_1^1 scheduled between 0 and 2 on R_1

>> Depth: 9, Number of states visited: 9
Critical demand peak:
R_4 between 5 and 8, Avg. expected demand: 0.44
Critical Operation: O_1^4, Avg. contrib.: 0.44
Using greedy reservation ordering heuristic:
O_1^4 scheduled between 7 and 10 on R_4

>> Depth: 10, Number of states visited: 10
Critical demand peak:
R_1 between 12 and 16, Avg. expected demand: 0.31
Critical Operation: O_2^3, Avg. contrib.: 0.31
Using greedy reservation ordering heuristic:
O_2^3 scheduled between 15 and 17 on R_1

>> Depth: 11, Number of states visited: 11
Critical demand peak:
R_3 between 13 and 15, Avg. expected demand: 0.14
Critical Operation: O_1^3, Avg. contrib.: 0.14
Using greedy reservation ordering heuristic:
O_1^3 scheduled between 14 and 15 on R_3

>> Depth: 12, Number of states visited: 12
Schedule Completed

Figure 3-9, concluded

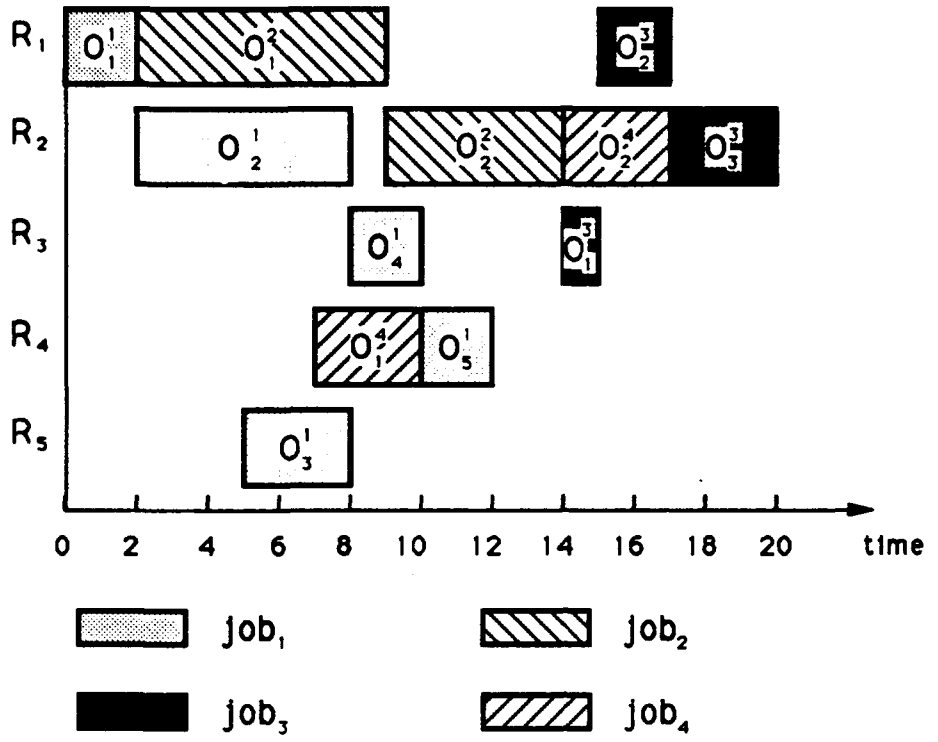


Figure 3-10: The final schedule produced by the micro-opportunistic scheduler.

question of how much flexibility/opportunism is really required to produce good schedules. Additional experimental results are reported in [Sadeh 91], including comparison against different macro-opportunistic schedulers that dynamically switch between a job-centered perspective and a resource-centered perspective, and an empirical evaluation of the impact of using biased demand profiles to identify important tradeoff operations.

3.4.1. Design of the Test Data

A set of 80 scheduling problems was randomly generated to cover a wide variety of scheduling conditions. Scheduling conditions were varied by adjusting a set of three parameters[17]: a parameter controlling the average due date of the jobs to be scheduled (**tardy factor**), a parameter controlling the variance of the job due dates (**due date range**), and a parameter controlling the **number of major bottleneck machines**.

These three scheduling parameters were set as follows:

- **Tardy Factor** (τ): this factor controlled the average tightness of the job due dates in the experiments. Given an optimistic estimate of the expected makespan of the problem, say M, the average job due date was set to $(1-\tau)M$, where τ is the

[17]These parameters or similar ones are commonly used in the scheduling literature to study a variety of shop floor situations [Srinivasan 71, Fisher 76, Ow 85, Morton et al. 88].

tardy factor[18]. If $\tau=0$ it might be possible to complete all jobs on time. As τ increases, the proportion of jobs that can still be completed on time decreases. In the scheduling problems that were generated, two values of the tardy factor were used: $\tau=0.2$ (loose average due date) and $\tau=0.4$ (tight average due date).

- **Due Date Range** (R): job due dates were randomly drawn from a uniform probability distribution $(1-\tau)M\,U(1-R/2, 1+R/2)$, where U(a,b) represents a uniform distribution between a and b, and R is the due date range. Two due date ranges were used: 1.6 (wide due date range) and 0.6 (tight due date range).

- **Number of Bottlenecks**: All problems involved five resources. In half of the problems, one out of the five resources was selected to be a major bottleneck, while the other half of the problems had two major bottlenecks.

Ten scheduling problems were randomly generated for each parameter combination, resulting in a total of 80 scheduling problems (10 problems x 2 tardy factors x 2 due date ranges x 2 bottleneck configurations). All experiments involved 20 jobs and 5 resources, for a total of 100 operations. All jobs had a linear process routing, and had to go through each machine exactly once. The order in which a job would go through the machines was randomly generated for each job, except for the bottleneck machines, which were always visited after a fixed number of operations (in order to further increase resource contention). In the case with one bottleneck resource, the bottleneck operation corresponded to the 4th operation (out of 5); in the case with two bottlenecks, the bottlenecks corresponded to the 2d and 5th operations of each job.

Job sizes (S_l): The size S_l of job j_l (i.e. the number of parts to be produced) was randomly drawn from a uniform distribution $U(1,7)$

Operation durations (du_i^l): Operation durations were correlated with job sizes. The duration of an operation O_i^l on a non-bottleneck resource was randomly drawn from a uniform distribution $S_l U(0.5, 1.5)$. In problems with a single bottleneck, the duration of an operation O_i^l requiring the bottleneck was set to $du_i^l = 2S_l$. In problems with two bottlenecks, the duration of the operation requiring the first bottleneck was set to $1.8S_l$ and that of the operation requiring the second bottleneck was set to $2S_l$.

Tardiness costs: the marginal tardiness cost, $tard_l$ of job j_l was randomly drawn from a uniform distribution $5U(1, 2S_l)$.

---

[18] $M = (n-1)\overline{du}_{R_{bott}} + \sum_{R=R_1}^{R_m} \overline{du}_R$, where n is the number of jobs, m the number of resources, R_{bott} the main bottleneck resource and $\overline{du}_R$ denotes the average duration of the operations requiring resource R. This estimate was first proposed in [Ow 85].

Inventory costs: Inventory costs were only introduced by the first operation in each job, namely operation O_1^l. These marginal inventory costs can be interpreted as the sum of the marginal holding costs and the interests on raw material costs. These costs are typically proportional to the size of the job. In these experiments, inv_1^l was simply set to S_l. Given that the mean of the marginal tardiness cost distribution is slightly larger than $5S_l$, this corresponds to a ratio of marginal tardiness cost over marginal inventory cost with a mean slightly larger than 5.

Earliest Acceptable Release Date/Latest Acceptable Completion Date(erd_l,lcd_l): In order to increase contention, all jobs were given the same earliest acceptable release date, erd_l=0. Since these experiments were designed to study the quality of the schedules produced by the micro-opportunistic approach, all jobs were given a non-constraining latest acceptable completion date[19], which was set to lcd_l=2M (where M is the optimistic estimate of the problem makespan used to set the average due date). Studies of problems with more constraining latest acceptable completion dates can be found in [Sadeh & Fox 90a, Sadeh & Fox 90b].

3.4.2. Comparison Against Four Dispatch Rules

A first set of 400 experiments was performed to compare MICRO-BOSS against four priority dispatch rules that are known to be particularly good at reducing tardiness [Vepsalainen & Morton 87]: the Weighted Shortest Processing Time (WSPT) rule, the Earliest Due Date (EDD) rule, the Slack per Remaining Processing Time (S/RPT) rule and the Weighted Cost OVER Time (WCOVERT) rule.

Figures 3-11 and 3-12 show the average costs of the schedules produced by each of the five scheduling techniques under all eight scheduling conditions. More detailed performance measures are provided in Appendix A, including weighted tardiness measures, weighted flowtime measures (in-process inventory), weighted earliness measures (finished-goods inventory), information relative to the search efficiency of the micro-opportunistic approach, etc.

As observed in other studies of these four dispatch rules [Vepsalainen & Morton 87], EDD and S/RPT tend to fare well on problems with a large due date range (R=1.6), while WSPT performs best on problems with a tight average due date and a narrow due date range. The

[19]Another reason for choosing non-constraining latest acceptable completion dates was that we wanted to compare the micro-opportunistic approach against some priority dispatch rules. These rules are unable to account for such constraints.

Job shops with 1 bottleneck					
	WSPT	EDD	S/RPT	WCOVERT	MICRO-OP
τ=0.2 R=1.6	**14,826** (3,607)	**10,265** (3,550)	**11,003** (3,974)	**10,888** (3,446)	**5,558** (2,353)
τ=0.2 R=0.6	**11,834** (3,362)	**12,308** (5,415)	**13,997** (6,929)	**11,996** (3,513)	**8,149** (3,101)
τ=0.4 R=1.6	**15,262** (3,833)	**11,609** (6,300)	**12,133** (5,653)	**11,643** (4,074)	**7,984** (4,557)
τ=0.4 R=0.6	**12,670** (3,854)	**15,044** (7,106)	**17,358** (7,845)	**13,229** (4,774)	**10,887** (5,651)

Figure 3-11: Comparison of the cost of the schedules produced by MICRO-BOSS, the WSPT, EDD, S/RPT and WCOVERT dispatch rules on job shop problems with one major bottleneck resource. Standard deviations appear between parentheses.

Job shops with 2 bottlenecks					
	WSPT	EDD	S/RPT	WCOVERT	MICRO-OP
τ=0.2 R=1.6	**18,459** (3,471)	**12,086** (2,632)	**12,827** (3,739)	**13,899** (3,261)	**9,435** (3,800)
τ=0.2 R=0.6	**13,398** (3,234)	**13,299** (2,305)	**15,314** (2,866)	**13,199** (3,043)	**10,302** (1,691)
τ=0.4 R=1.6	**19,628** (3,542)	**15,049** (4,042)	**16,119** (4,803)	**15,927** (4,074)	**12,346** (4,827)
τ=0.4 R=0.6	**15,146** (3,673)	**17,628** (3,465)	**20,441** (4,898)	**15,195** (3,535)	**13,051** (4,501)

Figure 3-12: Comparison of the cost of the schedules produced by MICRO-BOSS, the WSPT, EDD, S/RPT and WCOVERT dispatch rules on job shop problems with two major bottleneck resources. Standard deviations appear between parentheses.

WCOVERT rule usually fares well under all conditions, though it is often slightly outperformed by one of the three other priority rules. Remarkably enough, the micro-opportunistic approach outperforms all four dispatch rules under all eight scheduling conditions. The largest savings are achieved on the easiest problems, where there is more room for improvement. In particular, on the set of problems with a single bottleneck, loose average due dates ($\tau=0.2$), and wide due date ranges ($R=1.6$), the micro-opportunistic approach allows for savings of more than 45% over the best dispatch rule (EDD in this case). A more detailed analysis of the results (see Appendix A) shows that this saving is due to reductions of over 50% in in-process inventory as well as important reductions in finished-goods inventories. Even on the most difficult problems (two bottlenecks, tight average due date,$\tau=0.4$, and narrow due date range, $R=0.6$), where there is typically much less room for improvement, the micro-opportunistic approach still achieved savings of close to 15% over

the best dispatch rule (WSPT). The performance measures provided in Appendix A indicate that under all eight scheduling conditions, the micro-opportunistic approach produces schedules with an average weighted tardiness comparable to the best of these dispatch rules (which are among the best dispatch rules currently known to minimize this measure [Vepsalainen & Morton 87]), while allowing for important reductions in work-in-process, i.e. weighted flowtime, (often close to 50%) and finished-goods inventory, i.e. weighted earliness.

In most problems, the micro-opportunistic approach achieved a search efficiency of 100%, which was computed as the ratio of the number of operations to be scheduled over the number of search states that were visited. The worst set of problems with respect to search efficiency was the one with two bottlenecks, a tight average due date, and a wide due date range. Even for these problems, the micro-opportunistic scheduler was still able to achieve a search efficiency of 93%. In the set of 80 scheduling problems, the worst problem required to generate 174 search states (for 100 operations). On the average, the micro-opportunistic approach required around 20 minutes of CPU time to schedule a problem[20].

3.4.3. Varying the Granularity of the Micro-opportunistic Approach

The experiments reported above as well as other experiments comparing MICRO-BOSS against several macro-opportunistic schedulers confirmed our intuition that a more flexible approach to scheduling would allow for important savings both in tardiness and inventory costs. It is natural to ask ourselves how much of that flexibility is really necessary to achieve the savings observed in these experiments. In other words, how often is it necessary to revise the current strategy ? How often should the scheduler build new demand profiles, and switch its scheduling effort to a new more critical resource/time interval ? Given that in each search state, the micro-opportunistic scheduler spends most of its time updating demand profiles and looking for the most contented resource/time interval, important speedups could possibly be achieved, if schedules as good as those obtained by MICRO-BOSS could be built without performing these computations in each search state.

In order to study this issue, several opportunistic schedulers were built, which differed in the number, G, of operations that had to be scheduled before the scheduler looked for a new critical resource/time interval. The version of the scheduler with $G=1$ is the MICRO-BOSS scheduler studied throughout this chapter. For $G > 1$, the G operations with the largest contribution to the demand peak were scheduled before the scheduler would look for a new demand peak. The G critical operations were scheduled starting with those that relied most on the critical resource/time interval (i.e. starting with those with the largest contribution to the demand peak). For instance, in the example displayed in Figure 3-8, a scheduler with a

[20]Preliminary experiments in C seem to indicate that this time could be reduced to 1 minute.

granularity $G=3$ would first schedule O_2^1, then O_2^2, and finally O_3^3. The scheduler would then update the demand profiles and look for a new critical resource/time interval[21].

Job shops with 1 bottleneck			
	OPPORTUNISTIC, $G=3$	OPPORTUNISTIC, $G=2$	MICRO-BOSS, $G=1$
τ=0.2 R=1.6	**9,272** (4,909)	**7,163** (2,738)	**5,558** (2,353)
τ=0.2 R=0.6	**11,908** (4,625)	**11,409** (3,346)	**8,149** (3,101)
τ=0.4 R=1.6	**10,637** (5,591)	**10,347** (5,674)	**7,984** (4,557)
τ=0.4 R=0.6	**14,236** (5,058)	**13,228** (4,807)	**10,887** (5,651)

Figure 3-13: Comparison of the cost of the schedules produced by MICRO-BOSS and two coarser opportunistic schedulers that differ in the number of operations scheduled before the scheduling strategy can be revised. This number is referred to as the granularity of the opportunistic scheduler (G). The results in this table are for problems with one major bottleneck resource. Standard deviations appear between parentheses.

Job shops with 2 bottlenecks			
	OPPORTUNISTIC, $G=3$	OPPORTUNISTIC, $G=2$	MICRO-BOSS, $G=1$
τ=0.2 R=1.6	**11,655** (4,220)	**11,485** (3,961)	**9,435** (3,800)
τ=0.2 R=0.6	**15,720** (4,310)	**15,275** (3,596)	**10,302** (1,691)
τ=0.4 R=1.6	**15,471** (4,687)	**14,791** (5,679)	**12,346** (4,827)
τ=0.4 R=0.6	**18,701** (4,514)	**17,848** (4,480)	**13,051** (4,501)

Figure 3-14: Comparison of the cost of the schedules produced by MICRO-BOSS and two coarser opportunistic schedulers that differ in the number of operations scheduled before the scheduling strategy can be revised. This number is referred to as the granularity of the opportunistic scheduler (G). The results in this table are for problems with two major bottleneck resources. Standard deviations appear between parentheses.

The average schedule costs obtained for $G=1,2$ and 3 are displayed in Figures 3-13 and 3-14. These results indicate that the quality of the schedules consistently decreases when the granularity of the micro-opportunistic approach increases. These results corroborate

[21]The *mincost* functions are still updated in each search state since they are needed by the reservation ordering heuristic.

those of another study in which MICRO-BOSS was compared against a macro-opportunistic scheduler [Sadeh 91]. They suggest that all the flexibility of the micro-opportunistic scheduling approach is necessary to obtain the results reported earlier in the study. In other words critical resource/time intervals are highly dynamic, and it is critical to constantly follow their evolution in order to come up with quality schedules. Additional results provided in Appendix A, indicate that the deterioration observed for larger values of G is due to poorer performance both with respect to tardiness and inventory. They also show that increasing the granularity of the micro-opportunistic approach reduces search efficiency (especially for $G > 2$ and on problems with 2 bottlenecks). For $G=4$, the scheduler started having problems building schedules in less than 1,000 search states.

3.5. Conclusion

In this chapter, a micro-opportunistic approach to factory scheduling was described that closely monitors the evolution of bottlenecks during the construction of the schedule, and continuously redirects search towards the bottleneck that appears to be most critical. This approach differs from earlier opportunistic approaches described in [Ow & Smith 88] and [Collinot et al. 88] as it does not require scheduling an entire bottleneck resource or an entire job before revising the current scheduling strategy. This micro-opportunistic approach has been implemented in the context of the MICRO-BOSS factory scheduling system. Extensive experimental studies indicate that the flexibility of MICRO-BOSS enables it to outperform both Operations Research techniques such as the EDD, WSPT, S/RPT, and WCOVERT priority dispatch rules and other Artificial Intelligence techniques such as coarser opportunistic schedulers under a wide variety of scheduling conditions.

3.6. Appendix A: Additional Performance Measures

This appendix provides additional performance measures for the experiments described in section 4. Performances are reported along the following metrics:

- **Average weighted tardiness**:

$$\frac{\sum_{l} tard_l \times Max(0, C_l - dd_l)}{\sum_{l} tard_l}$$

See Figure 3-15.

- **Average weighted flowtime**:

$$\frac{\sum_{l} inv_1^l \times (C_l - st_1^l)}{\sum_{l} inv_1^l}$$

This measure corresponds to the average in-process inventory of the schedule (or work-in-process). See Figure 3-16.

- **Average weighted earliness**:

$$\frac{\sum_{l} inv_1^l \times Max(0, dd_l - C_l)}{\sum_{l} inv_1^l}$$

This measure corresponds to the average finished-goods inventory of the schedule. See Figure 3-17

- **Search efficiency**: the number of operations to be scheduled (100) divided by the number of search states generated. Search efficiency is only reported for the opportunistic schedulers, since priority dispatch rules are one-pass procedures that never backtrack.

Problem Sets			
Number of Bottlenecks	R	τ	Problem Set
1	0.2	1.6	1
1	0.2	0.6	2
1	0.4	1.6	3
1	0.4	0.6	4
2	0.2	1.6	5
2	0.2	0.6	6
2	0.4	1.6	7
2	0.4	0.6	8

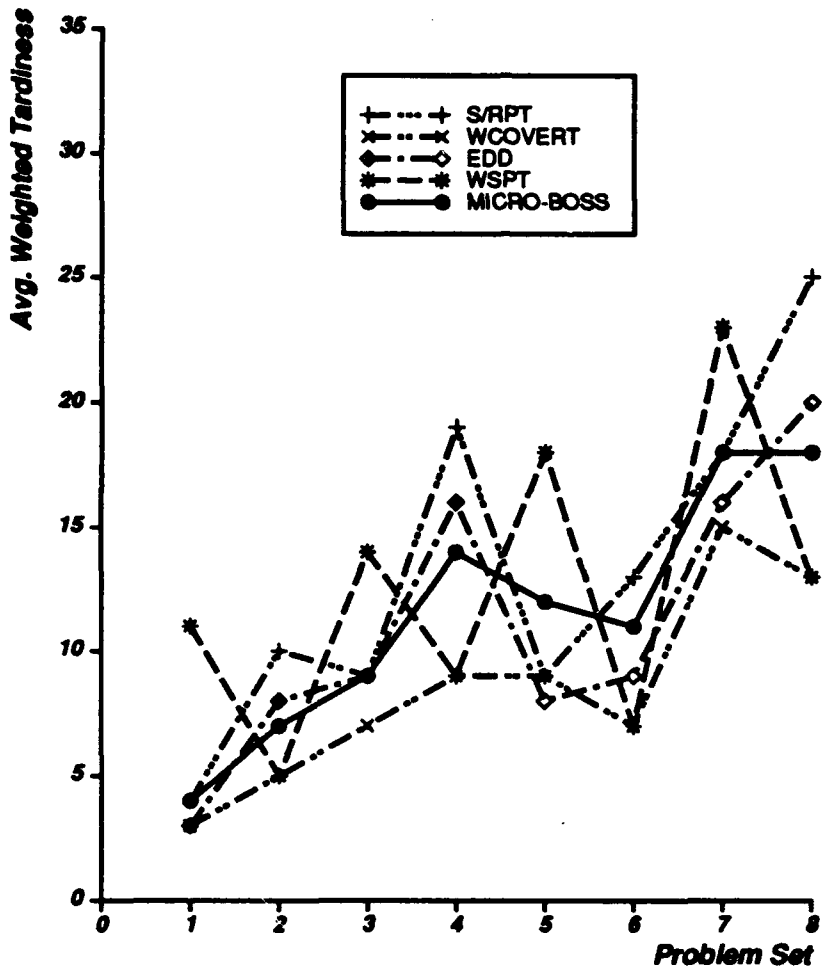


Figure 3-15: Weighted tardiness performance.

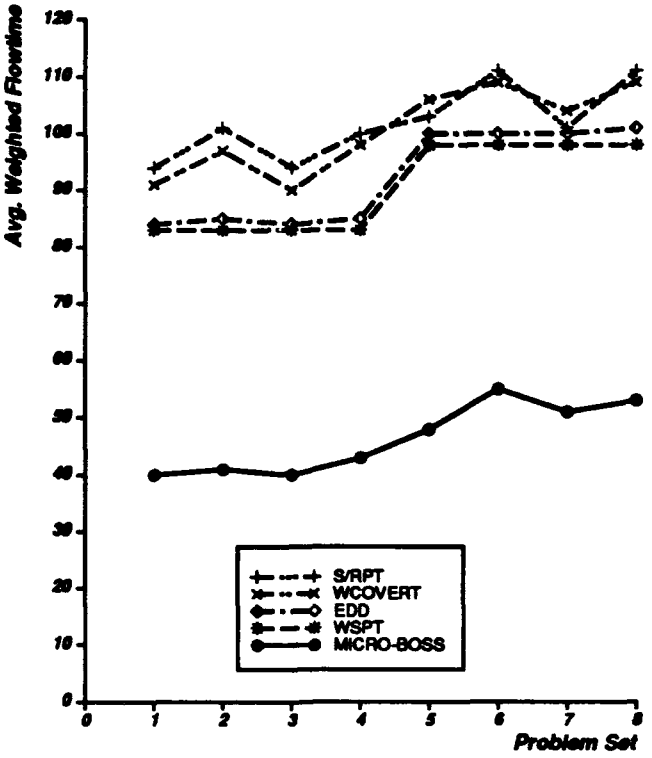


Figure 3-16: Weighted flowtime (i.e. work-in-process) performance.

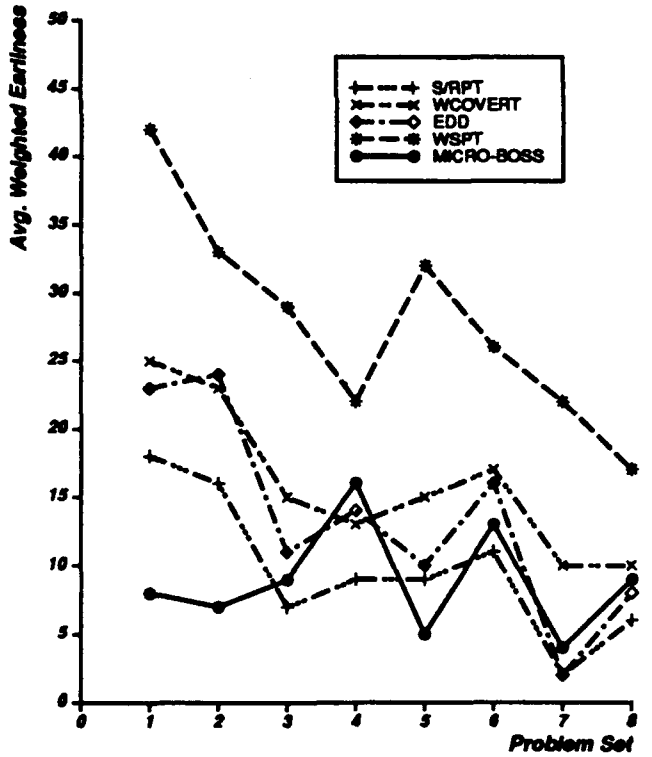


Figure 3-17: Weighted earliness (i.e. finished-goods inventory) performance.

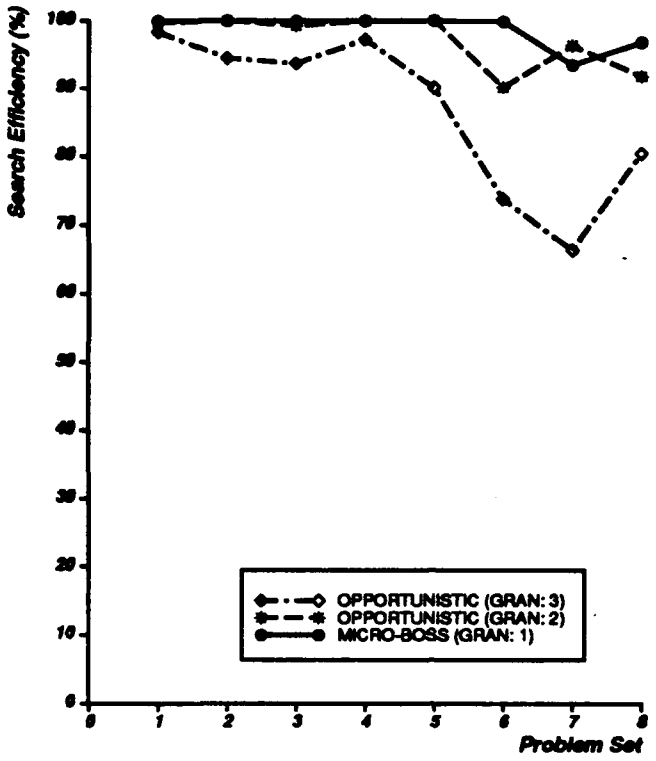


Figure 3-18: Search Efficiency.

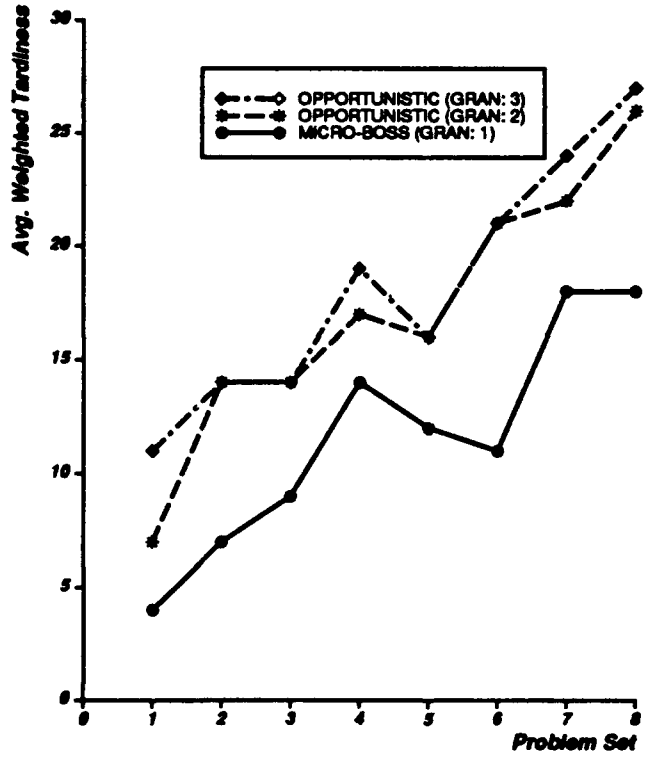


Figure 3-19: Weighted Tardiness Performance for MICRO-BOSS and two coarser opportunistic schedulers.

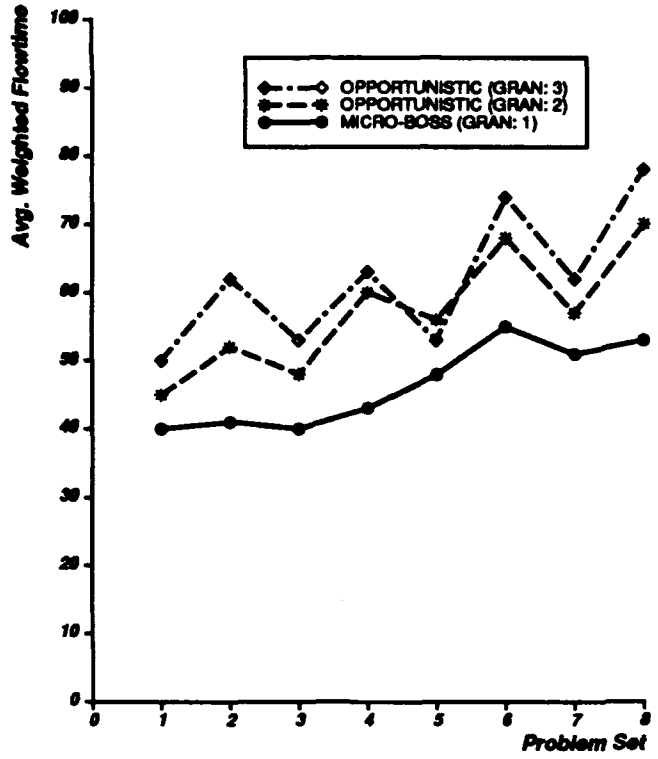


Figure 3-20: Weighted Flowtime Performance for MICRO-BOSS and two coarser opportunistic schedulers.

Chapter 4
Distributed Scheduling

Summary

In this chapter we present a model of decentralized problem solving, called Distributed Constrained Heuristic Search (DCHS) that provides both structure and focus in individual agent search spaces so as to optimize decisions in the global space. The model achieves this by integrating decentralized constraint satisfaction and heuristic search. It is a formalism suitable for describing a large set of DAI problems. We introduce the notion of textures that allow agents to operate in an asynchronous concurrent manner. The employment of textures coupled with distributed asynchronous backjumping (DAB), a type of distributed dependency-directed backtracking that we have developed, enables agents to instantiate variables in such a way as to minimize backtracking. We have experimentally tested our approach in the domain of decentralized job-shop scheduling. A formulation of distributed job-shop scheduling as a DCHS is presented as well as experimental results.

4.1. Introduction

In this chapter, we present a framework for formalizing a set of DAI problems by extending Constrained Heuristic Search (CHS) [Fox et al. 89] to multi-agent environments. A *distributed Constrained Heuristic Search (DCHS* problem) is a CHS problem where the solution is the result of cooperative multi-agent problem solving. The CHS problem solving paradigm addresses a subclass of problems that can be solved through state space search. Similarly, DCHS addresses a subclass of DAI problems that can be solved through distributed search. This methodological commitment is consistent with other research that formulates DAI problems in terms of search. We are currently engaged in investigating and further developing the CHS model in both single and multi-agent settings.

The CHS problem solving model provides both structure and focus to search in the problem space. The model achieves this by combining the process of constraint satisfaction (CSP) with heuristic search (HS). The resulting model both reduces search complexity and provides a more formal explanation of the nature and power of heuristics in problem solving. Although both CSP and HS have been extensively studied for single agent problem solving,

with the notable exception of [Yokoo et al. 90], there have been no attempts at studying these formalisms in a multi-agent setting. However, formal investigations of distributed CSP and HS problem solving models is very important because, as has been pointed out in [Yokoo et al. 90], (1) Various DAI problems can be formulated as distributed CSP or distributed HS problems (e.g., [Lesser 90, Conry et al. 88, Kuwabara & Lesser 89]) and (2) distributed CSP and HS models can provide formal frameworks for investigating various DAI issues and methodologies, such as decision making coherence (e.g., [Durfee 87]) and organizational redesign (e.g., [Ishida et al. 90]).

CHS integrates the synthetic capabilities of heuristics search with the structural characteristics of constraint satisfaction techniques. Constraint satisfaction algorithms are viewed as taking giant steps, not creating new objects, but reducing the entire space of objects to a satisficing set. (This assumes the ability to enumerate a set of objects from which to choose.) On the other hand, search techniques can be synthetic in that they incrementally construct a solution as part of the search process. In formulating the framework for distributed CHS, we are concerned with the principles behind how knowledge can be used to structure and guide asynchronous distributed search in the problem space of individual agents so as to optimize overall problem solving of the multi-agent system[22].

In this chapter, we first review the elements of the CHS problem solving model. We then introduce the definition of DCHS and give the general process by which agents perform asynchronous DCHS. Section 4 formulates the distributed job-shop scheduling problem as a DCHS. The variable and value ordering heuristics that have been developed are presented in detail. Section 4.3 presents distributed asynchronous backjumping (DAB). Section 5 presents the communication protocol that allows concurrent asynchronous problem solving by the agents. Section 6 presents experimental results and section 7 presents concluding remarks.

4.2. Overview of Constrained Heuristic Search

CHS augments the definition of a problem space [Newell & Simon 76], composed of states, operators and an evaluation function, by refining a state to include:

1. **Problem Topology**: Provides a structural characterization of a problem.
2. **Problem Textures**: Provide measures of a problem topology that allows search to be focused in a way that reduces backtracking.
3. **Problem Objective**: Defines an objective function for rating alternative solutions that satisfy a goal description.

This model allows us to (1) view problem solving as constraint optimization, thus taking

[22] A problem space is composed of an initial state that defines the problem's initial conditions, a set of operators that generate new states and an evaluation function that identifies solution states.

advantage of optimization techniques, (2), incorporate heuristic search, thus allowing the dynamic modification of the constraint model, and (3) extend constraint satisfaction to the larger class of optimization problems.

We define problem topology as a graph G, composed of vertices V and edges E:

$V = N \cup C \cup S$

where

N is a set of variables $\{n_1, n_2, ..., n_m\}$

C is a set of constraints $\{c_1, c_2, ..., c_n\}$

S is a set of satisfiability specifications $\{s_1, s_2, ..., s_o\}$

Each variable in N may be a vector of variables whose domains may be finite/infinite and continuous/discrete. Constraints are n-ary predicates over variables vertices. A constraint predicate is true iff the instantiations of these variable vertices are compatible with each other. A satisfiability specification vertex groups constraints into sets of type AND, OR, or XOR. An XOR satisfaction set denotes that only one constraint in the set must be satisfied. Edges link constraint vertices to variable vertices, and satisfiability specifications to constraints. Finding a solution to a CHS problem consists in finding an assignment of values to all variables that satisfies all constraints and all satisfiability specifications.

We distinguish between two types of problem topologies:

Definition 1: A *completely structured problem* is one in which all non-redundant vertices and edges are known a priori.

This is true of all CSP formulations.

Definition 2: A *partially structured problem* is one in which not all non-redundant vertices and edges are known prior to problem solving.

This definition tends to be true of problems in which synthesis is performed resulting in new variables and constraints (e.g. the generation of new subgoals during the planning process).

Operators in CHS have many roles: refining the problem by adding new variable and constraint vertices, reducing the number of solutions by reducing the domains of variables (e.g., assigning a value to a variable vertex), or reformulating the problem by relaxing constraints or omitting constraints and/or variables.

The general CSP is a well-known NP-complete problem [Garey & Johnson 79]. There are however classes of CSPs that do not belong to NP, and for which efficient algorithms exist. The CHS methodology is meant for those CSPs for which there is no efficient algorithm. A general paradigm for solving these problems consists in using Backtrack Search (BT) [Golomb & Baumert 65, Bitner & Reingold 75]. BT is an enumerative technique that incrementally builds a solution by instantiating one variable after another. Each time a new variable is instantiated, a new search state is created that corresponds to a more complete partial solution. If, in the process of building a solution, BT generates a partial solution that

it cannot complete (because of constraint incompatibility), it has to undo one or several earlier decisions. Partial solutions that cannot be completed are often referred to as deadend states (in the search space). Because the general CSP is NP-complete, BT may require exponential time in the *worst-case*. CHS provides a methodology to reduce the *average* complexity of BT by interleaving search with *local constraint propagation* and the computation of *texture-based heuristics*.

Local constraint propagation techniques are used to prune alternatives that have become impossible due to earlier decisions made to reach the current search state. By propagating the effects of earlier commitments as soon as possible, CHS reduces the chances of making decisions that are incompatible with these earlier commitments [Mackworth & Freuder 85].

Typically, pruning the search space can only be done efficiently on a local basis [Nadel 88]. Hence local constraint propagation techniques are not sufficient to guarantee backtrack-free search. In order to avoid backtracking as much as possible as well as reduce its impact when it cannot be avoided, we need techniques for focusing the problem solver's attention opportunistically to promising decisions. In CHS, for search to be well focused, that is to decide where in the problem topology an operator is to be applied, there must be features of the topology that differentiate one subgraph from another, and these features must be related to the goals of the problem. These features take the form of different types of constraint interactions. CHS analyzes the pruned problem space in order to determine critical variables, promising values for these variables, promising search states to backtrack to, etc. The results of this analysis are summarized in a set of textures that characterize different types of constraint interactions in the search space. We have identified and are experimenting with seven such problem textures: Value Goodness, Constraint tightness, Variable Goodness, Variable Tightness, Constraint Reliance, and Variable Tightness with respect to a set of constraints [Fox et al. 89]. These textures are operationalized by a set of heuristics to decide which variable to instantiate next (so-called variable ordering heuristics), which value to assign to a variable (so-called value ordering heuristics), which assignment to undo in order to recover from a deadend, etc. These textures generalize the notion of constraint satisfiability or looseness defined by [Nadel 86] and apply to both CHSs (and CSPs) with discrete and continuous variables. We have generalized these textures so they can be used in Distributed CHS.

4.3. Distributed CHS

A distributed CHS problem is a problem where the variables are distributed among a set of agents. Each agent is responsible for a set of variables and their instantiation. Constraints and satisfiability conditions exist among variables under the jurisdiction of different agents. The instantiation of variables must satisfy these constraints and satisfiability specifications. We say that a distributed CHS problem is solved iff an assignment is found of values to all variables of all agents such that all constraints and satisfiability specifications are simultaneously satisfied.

Distributed DCHS is a process carried out by a group of agents each of which has (a) limited knowledge of the environment, (b) limited knowledge of the constraints and requirements of other agents, and (c) limited number and amount of resources that are required to produce a system solution. Global system solutions are arrived at by interleaving of local computations and information exchange among the agents. There is no single agent with a global system view. In such an environment, DCHS is an incremental process. Agents make local decisions about assignments of values to particular variables at particular times during problem solving and a complete solution is formed by incrementally merging partial solutions.

The Distributed CHS problem has the following characteristics:

- The global system goal is to assign in a distributed fashion a set of values to a set of variables so that all constraints are satisfied and backtracking is minimized.
- To achieve global solutions, agents have to make consistent variable instantiations. In our model, the variable instantiations are made concurrently and asynchronously. Each agent instantiates the variables under its control and communicates the variable values to agents that need to know these values.
- Due to limited communication bandwidth, it is not possible for agents to exchange a complete set of specific constraints.
- Because each agent has incomplete information, it is in general impossible for each agent to assign consistent values to variables such that constraints are satisfied using only local information.
- Local computations could produce inconsistent value assignments, ie value assignments that lead to constraint violations. When this happens one or more agent(s) has(have) to backtrack and try again. Backtracking can have major ripple effects on the network since it may invalidate value assignments that other agents have made. Moreover, asynchronous backtracking runs the risk of computing irrelevant action based on obsolete information.

There are two remarks in order here with respect to backtracking. First, the standard chronological backtracking [Mackworth & Freuder 85] instantiates variables in some sequential order. In general backtracking search is exponential in the worst-case. The situation is worse in distributed asynchronous backtracking. One way to increase backtracking efficiency is through various forms of dependency-directed backtracking

[deKleer 87, Gaschnig 79, Dechter 89]. We have implemented a variant of dependency directed backtracking for distributed, asynchronous problem solving, called distributed asynchronous backjumping (DAB) that substantially reduces distributed search (see sections 4.3 and 6).

Second, both empirical and analytical studies in the CSP literature show that it is possible to reduce backtracking by properly sequencing the order in which variables are instantiated. As a consequence, a lot of research in CSP has concentrated on developing good variable and value ordering heuristics. A good variable ordering heuristic is to instantiate the variables that are most tightly constrained. This way, the system avoids investing a lot of time building partial solutions that cannot be completed because some difficult variables had not yet been instantiated. A good value ordering heuristic to reduce backtracking is a least-constraining one, namely one that leaves as much room as possible to other variables and their values so that the current partial solution can be completed without backtracking.

Because of the incomplete and asynchronously changing information in the distributed case, the agents at each stage of problem solving require additional constraint instantiation guidance in the form of mechanisms to (a) predict and evaluate the impact of their decisions on global system goals, (b) form expectations and predictions about the constraints of other agents, and (c) help focus the attention of the agents on particular parts of their search space asynchronously and opportunistically. Having good predictive measures is very important in the distributed case because:

1. asynchronous backtracking is more costly when it involves many agents (ripple effects)
2. since agents operate asynchronously and concurrently, they may be called upon to estimate value assignments that they may not need to consider until they have made many other instantiations
3. since the agents operate asynchronously and concurrently, they have to predict and take into consideration in their local decision-making the *future* needs of other agents that are in an earlier stage of problem solving
4. since communication is costly the predictive measures must be robust/predictive over many decisions

The textures that have been identified in the previous section have been generalized to work for DCHS. Our hypothesis is that these textures are good predictive measures of the impact of local decisions on system goals and express expectations of the difficulty of satisfying constraints at various parts of the search space of agents. We have operationalized these textures into a set of heuristics that direct search in individual agent spaces.

Each agent's DCHS asynchronous problem solving contains the following steps:

- An initial state is defined composed of a problem topology,
- Constraint propagation is performed within the state,
- Texture measures and the problem objective are evaluated for the state's topology,

- Operators are matched against the state's topology, and
- A variable node/operator pair is selected and the operator is applied.

The application of an operator results in either adding structure to the topology, further restricting the domain of a variable, or reformulating the problem (e.g., relaxation). The results of operator application are then propagated both within the agent (local constraint propagation step) and across agents (a constraint communication step followed by a local constraint propagation step). If an inconsistency (i.e. constraint violation) is detected during propagation, the agent employs DAB. Otherwise the agent moves on and looks for a new variable node/operator pair to apply. The process goes on until all variables of all agents have been assigned consistent values that satisfy all constraints and satisfiability specifications.

By allowing search to be performed concurrently and asynchronously by several problem solving agents, we introduce two groups of tradeoffs:

1. A group of *design tradeoffs*, which is found under one form or another in the design of most distributed systems. In order to maintain search efficiency at a level comparable to that in the centralized setting (and hence achieve higher overall responsiveness), agents in the distributed system need to maintain a high degree of global system awareness. Within the centralized CHS setting, system awareness is achieved via local constraint propagation and texture computation. Unfortunately the limited communication bandwidth of a distributed system restricts the amount of information that can be passed between portions of the search space that are under the control of different agents. In other words the limited communication bandwidth generally prevents agents from attaining a level of awareness similar to that in a centralized system. As a consequence there is a tradeoff between the amount of distribution and the ability of the system to maintain a search efficiency that entails an overall increase in system responsiveness. For instance, in the factory scheduling domain, it is a good practice to start scheduling bottleneck resources first. Depending on the number of scheduling agents, and the way jobs are distributed between these agents, it is more or less difficult for the system as a whole to identify the bottleneck resources, and coordinate the construction of the schedule around the scheduling of these bottlenecks. Most distributed systems have to deal with this tradeoff under one form or another, as it generally determines the proper level of distribution in the system for a given bandwidth (or the necessary bandwidth given a predetermined number of agents), proper ways to partition the search space among a set of agents, etc.

2. A group of *tactical tradeoffs*: Given a specific distributed system with a predetermined number of agents, a fixed communication bandwidth, and a partitioned search space, whose portions have been preallocated to different agents in the system, there is still a large number of tactical parameters that one can play with in order to increase overall system responsiveness. In the distributed CHS setting, these parameters include the selection of the local constraint propagation mechanisms, the texture-based heuristics, and the synchronization protocols. All the tradeoffs influencing the selection of these parameters in the centralized setting are still present, but they have changed: they have become more complex due to the limited bandwidth of the system and the ability of the agents to perform some computations asynchronously. Consider for instance the relation between variable and value ordering

heuristics. When several agents are allowed to asynchronously instantiate variables, they lose some of the benefits of a good variable ordering. This problem can be remedied in two ways: either by using a less constraining value ordering heuristic (this would be equivalent to relaxing the due dates of jobs requiring a bottleneck resource) and/or by introducing some synchronization between the agents in order to enforce some degree of system-wide variable ordering (e.g. enforcing that bottlenecks are scheduled first by properly synchronizing the scheduling agents). Clearly both approaches have their inconveniences and put additional strains on the communication bandwidth. A less constraining value ordering heuristic can only be allowed via additional computational efforts locally and/or via additional inter-agent communication. Worse, using less constraining values typically translates into poorer solutions. On the other hand, enforcing synchronization between the agents can only be achieved via additional inter-agent communication, and restricts the amount of concurrent processing in the system.

In this chapter, we assume a distributed architecture and focus on the study of some of the tactical tradeoffs discussed above. While the study of some of the design tradeoffs identified above has been given some attention in the literature (e.g. [Fox 81, Cammarata et al. 83, Durfee et al. 85]), we do not know of any prior study of the tactical tradeoffs discussed in this chapter, namely those involved in distributing the CSP/CHS paradigm. The next section demonstrates the application of the distributed CHS model to the problem of distributed job shop scheduling.

4.4. Distributed CHS Job-Shop Scheduling

Factory scheduling has been the subject of intense investigation by both Operations Research and AI communities (e.g., [Baker 74, Rinnooy Kan 76, Grave 87, Fox 83, Smith et al. 86]). With few exceptions [Parunak et al. 86, Smith & Hynynen 87], there has been almost no research in distributed scheduling. On the other hand, the Distributed AI community has focused its attention primarily on problems where agents contend only for computational resources, such as computer time and communication bandwidth (e.g., [Cammarata et al. 83, Durfee 87]). In most real world situations, however, allocation of (non-computational) resources that are needed by an agent to carry out actions in a plan is of central concern. Hence, approaches and mechanisms are needed to allow for cooperative distributed resource allocation over time (i.e., distributed scheduling of resources).

The distributed job shop scheduling problem is viewed as a distributed CHS problem where each activity is an aggregate variable whose values are reservations. Our activity-based approach to job-shop scheduling relies on the combination of local constraint propagation techniques with texture-based heuristic search. A reservation consisting of a start time and a set of resources to be allocated to the activity. Each activity constitutes a variable vertex in the problem topology. Activity precedence constraints are binary constraints represented by constraint vertices connected to two activity variable vertices. A capacity constraint vertex is

associated with each resource and connected to all the variable vertices representing activities that can possibly use the resource. Each capacity constraint ensures that the corresponding resource will not be allocated to more than one activity at any given time.

Formally, we will say that we have a set of scheduling agents, $\Gamma=\{\alpha, \beta, ...\}$. Each agent α is responsible for the scheduling of a set of orders $O^{\alpha}=\{o_1^{\alpha},...,o_{N_\alpha}^{\alpha}\}$. Each order o_l^{α} consists of a set of activities $A^{l\alpha}=\{A_1^{l\alpha},...,A_{n_{l\alpha}}^{l\alpha}\}$ to be scheduled according to a process plan (i.e. process routing) which specifies a partial ordering among these activities (e.g. $A_p^{l\alpha}$ BEFORE $A_q^{l\alpha}$). Additionally an order has a release date and a latest acceptable completion date, which may actually be later than the ideal due date. Each activity $A_k^{l\alpha}$ also requires one or several resources $R_{ki}^{l\alpha}$ $(1 \le i \le p_k^{l\alpha})$, for each of which there may be one or several alternatives (i.e. substitutable resources) $R_{kij}^{l\alpha}$ $(1 \le j \le q_{ki}^{l\alpha})$. There is a finite number of resources available in the system. Some resources are only required by one agent, and are said to be *local* to that agent. Other resources are *shared*, in the sense that they may be allocated to different agents at different times.

We distinguish between two types of constraints: activity precedence constraints and capacity constraints. The activity precedence constraints together with the order release dates and latest acceptable completion dates restrict the set of acceptable start times of each activity. The capacity constraints restrict the number of activities that a resource can be allocated to at any moment in time to the capacity of that resource. For the sake of simplicity, we only consider resources with unary capacity in this chapter. Typically the limited capacity of the resources induces interactions between orders competing for the possession of the same resource at the same time. These interactions can take place either between the order of a same agent or between the orders of different agents.

With each activity, we associate preference functions that map each possible start time and each possible resource alternative onto a preference. These preferences [Fox 83, Sadeh & Fox 88] arise from global organizational goals such as reducing order tardiness (i.e. meeting due dates), reducing order earliness (i.e. finished goods inventory), reducing order flowtime (i.e. in-process inventory), using accurate machines, performing some activities during some shifts rather than others, etc. In the cooperative setting assumed in this chapter, the sum of these preferences over all the agents in the system and over all the activities to be scheduled by each of these agents defines a common objective function to be optimized. The sum of these preferences over all the activities under the responsibility of a single agent can be seen as the agent's local view of the global objective function. In other words, the global objective function is not known by any single agent. Furthermore, because they compete for a set of shared resources, it is not sufficient for an agent to try to optimize his own local preferences. Instead, agents need to consider the preferences of other agents when they schedule their activities.

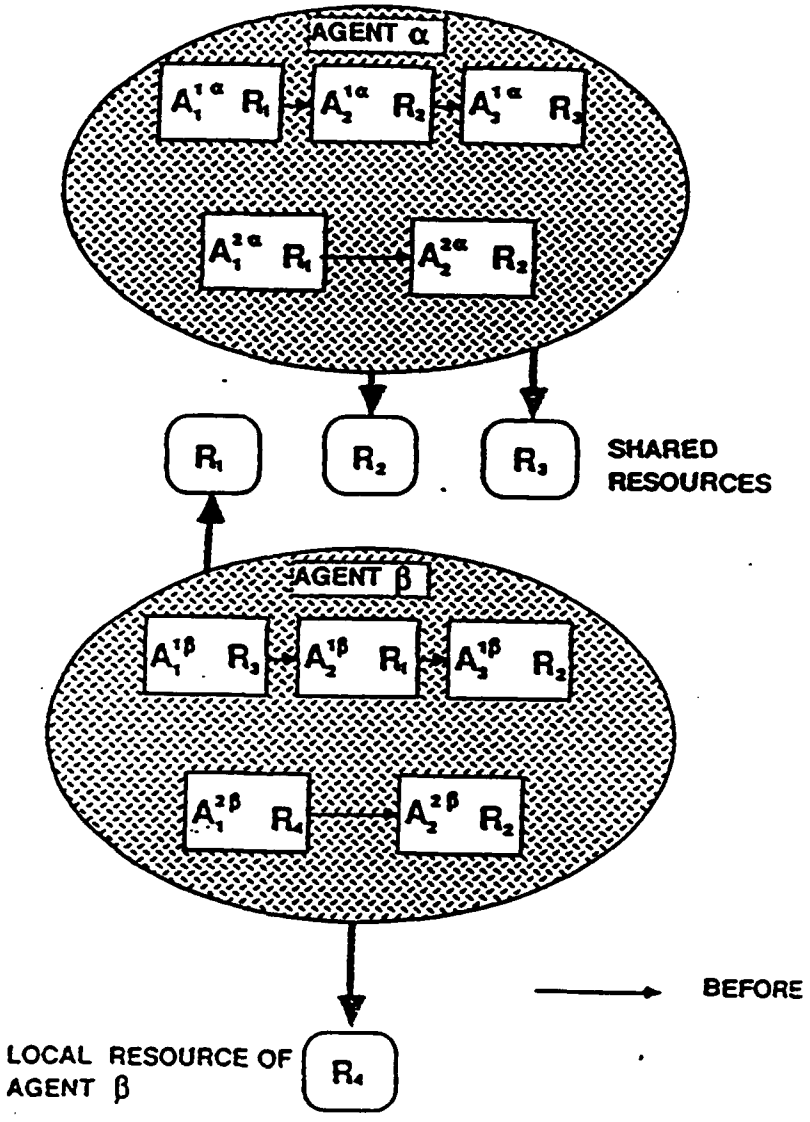


Figure 4-1: A simple problem with 2 agents, 4 orders, and 4 resources.

Figure 4-1 displays a simple example with 2 agents: α, and β. Agent α has two orders: o_1^{α} and o_2^{α}. Agent β also has two orders: o_1^{β} and o_2^{β}. The activities required by each order are specified along with their resource requirements. For instance,order o_1^{α} has a process plan with 3 activities: $A_1^{1\alpha}$ (which requires resource R_1), $A_2^{1\alpha}$ (which requires resource R_2), and $A_3^{1\alpha}$ (which requires resource R_3). The arrows between the activities represent the precedence constraints (e.g. $A_1^{1\alpha}$ has to be performed before $A_2^{1\alpha}$,). In this simple example, each activity has only one resource requirement, for each of which there is only one alternative (e.g. $R_{11}^{1\alpha}=R_1$). In other words the only variables in this problem are the activity

start times. We further assume that time has been discretized with a granularity of 1, that all the activities have the same duration, namely 3 time units, that all orders are released at time 0 and have to be completed by time 15[23]. Resources R_1, R_2, and R_3 are shared in this example, since they are required by both agent α and agent β. On the other hand, resource R_4 is local to agent β. Notice that resource R_2 is the only resource to be required by 4 activities (one in each order). All other resources are required by fewer activities. Later we will see that this resource is the main bottleneck of the problem. This example will also be used to illustrate how agents α and β coordinate to identify this bottleneck and avoid making conflicting reservations when scheduling operations requiring a shared resource.

In our model we view each activity $A_k^{l\alpha}$ as an aggregate *variable* (or vector of variables). A *value* is a reservation for an activity. It consists of a start time and a set of resources for that activity (i.e. one resource $R_{kij}^{l\alpha}$ for each resource requirement $R_{ki}^{l\alpha}$ of $A_k^{l\alpha}$, $1 \leq i \leq p_k^{l\alpha}$).

Each agent asynchronously builds a schedule for the orders he has been assigned. This is done incrementally by iteratively selecting an activity to be scheduled and a reservation for that activity. Each time a new activity is scheduled, new constraints are added to the system that reflect the new activity reservation. These new constraints are propagated both within the agent (local constraint propagation step) and across agents (a constraint communication step followed by a local constraint propagation step). If an inconsistency (i.e. constraint violation) is detected during propagation, the system backtracks. Otherwise the agent moves on and looks for a new activity to schedule and a reservation for that activity. The process goes on until all activities have been successfully scheduled.

If an agent could always make sure that the reservation that he is going to assign to an activity will not result in some constraint violation forcing him or other agents to undo earlier decisions, scheduling could be performed without backtracking. Because scheduling is NP-hard, it is commonly believed that such look-ahead cannot be performed efficiently. Instead the constraint propagation mechanism used in our system is the one described in [LePape & Smith 87]. For sake of efficiency, this mechanism does not attempt to guarantee total consistency, but instead looks for two types of inconsistencies that are easy to spot: violation of precedence constraints within an order, and violation of capacity constraints between a group of scheduled activities and one unscheduled activity. The conflicts that are not immediately detected by this mechanism correspond to violations of capacity constraints between several unscheduled activities[24]. This is because this last type of conflict appears to be more difficult to detect in general (possibly requiring exponential time). Under these

[23]None of these assumptions is required by our scheduling system. They simply make the example easier to understand.

[24]These conflicts are only detected later on, typically when all but one of the activities involved in the conflict have been scheduled.

conditions, a reservation assigned by an agent to an activity may force other agents or the agent himself to backtrack later on[25]. Consequently, it is important to focus search in a way that reduces the chances of having to backtrack and minimizes the work to be undone when backtracking occurs. This is accomplished via two techniques, known as *variable* (i.e. activity) and *value* (i.e. reservation) ordering heuristics.

The variable ordering heuristic assigns a *criticality measure* to each unscheduled activity; *the activity with the highest criticality is scheduled first.* The criticality measure approximates the likelihood that the activity will be involved in a conflict. The only conflicts that are accounted for in this measure are the ones that cannot be prevented by the constraint propagation mechanism. By scheduling his most critical activity first, an agent reduces his chances of wasting time building partial schedules that cannot be completed (i.e. it will reduce both the frequency and the damage of backtracking). The value ordering heuristic attempts to leave enough options open to the activities that have not yet been scheduled in order to reduce the chances of backtracking. This is done by assigning a *goodness* measure to each possible reservation of the activity to be scheduled. Both activity criticality and value goodness are examples of *texture measures*. The next two paragraphs briefly describe both of these measures[26].

4.4.1. Variable Ordering Scheduling Heuristic

As was just pointed out earlier, there are situations with insufficient capacity that may go undetected for a while by the constraint propagation technique, thereby causing the system to backtrack later on. Accordingly a critical activity is one whose resource requirements are likely to conflict with the resource requirements of other activities. [Sadeh & Fox 88, Sadeh & Fox 89] describes a technique to identify such activities. The technique starts by building for each unscheduled activity a probabilistic *activity demand*. An activity $A_k^{l\alpha}$'s demand for a resource $R_{kij}^{l\alpha}$ at time t is determined by the ratio of reservations that remain possible for $A_k^{l\alpha}$ and require using $R_{kij}^{l\alpha}$ at time t over the total number of reservations that remain possible for $A_k^{l\alpha}$. Clearly activities with many possible start times and resource reservations tend to have smaller demands at any moment in time, while activities with fewer possible reservations tend to have higher ones.

In a second step, each agent aggregates his activity demands as a function of time, thereby obtaining his *agent demand*. This demand reflects the need of the agent for a resource over

[25]This is already the case in the centralized version of the scheduling problem. Because of the additional cost of communication it is even more so in the distributed case.

[26]For a more complete description of these measures, the reader is referred to [Sadeh & Fox 90a].

time, given the activities that he still needs to schedule[28].

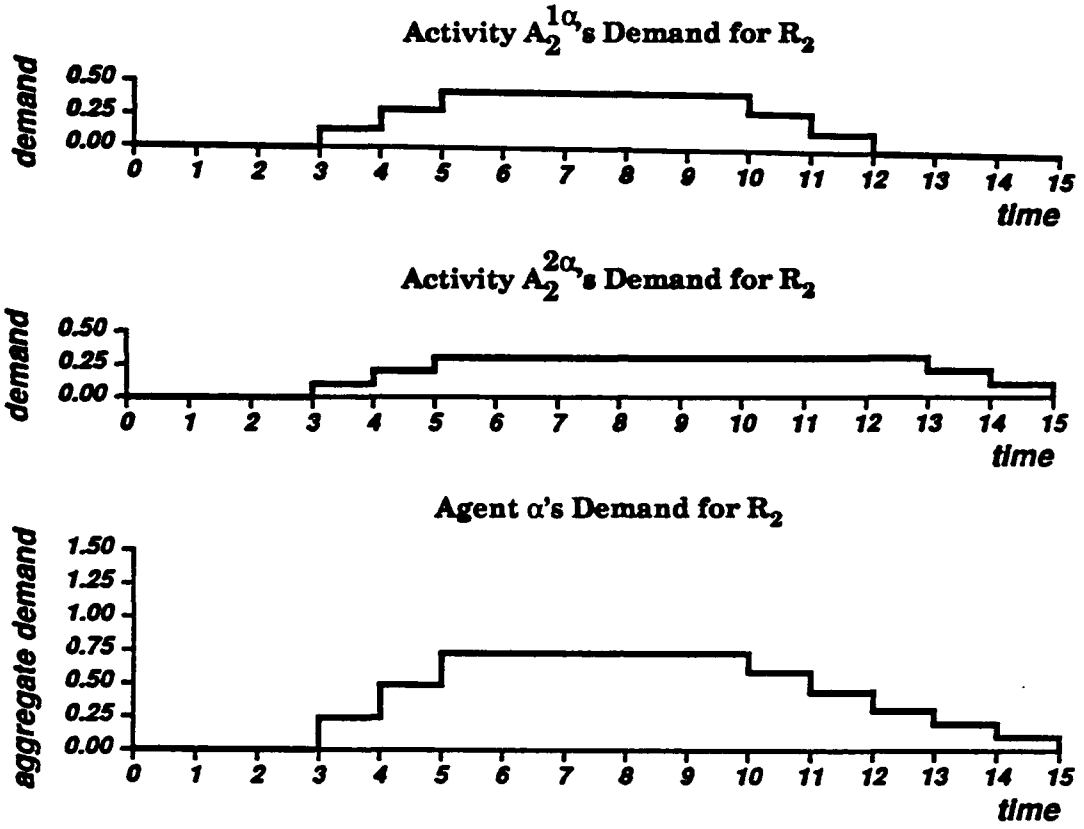


Figure 4-2: Building agent α's demand for resource R_2.

Figures 4-2 and 4-3 illustrate the process by which agent α and β compute their total (agent) demands for resource R_2. For instance, agent α has two activities requiring R_2: $A_2^{1\alpha}$ and $A_2^{2\alpha}$. In a first phase, the probabilistic demand of each of these two activities was computed by agent α, as illustrated in Figure 4-2. The computation is done as follows: Each agent calculates for each activity the set of remaining possible earliest-start-times and latest-start-times after existing reservations have already been taken into account. For example, activity $A_2^{1\alpha}$ has earliest-start-time of 3 and latest-start-time of 9 (to allow for scheduling of its preceding activity $A_1^{1\alpha}$, and its succeeding activity $A_3^{1\alpha}$). Similarly, activity $A_2^{2\alpha}$ has earliest-start-time of 3 and latest-start-time of 12 (to allow for scheduling its preceding activity $A_1^{2\alpha}$). Assuming that no reservations have been made yet, activity $A_2^{1\alpha}$ has 7 possible reservations on R_2 whereas activity $A_2^{2\alpha}$ has 10. Thus, the probabilistic demand for resource R_2 of $A_2^{1\alpha}$ for time interval [3, 4] is 1/7, for time interval [4, 5] it is 1/7 +

[28]Notice that, an agent's demand at some time t for a resource is obtained by simply summing the demands of all his unscheduled activities at time t. Because these probabilities do not account for limited resource capacities, their sum may actually get larger than 1.

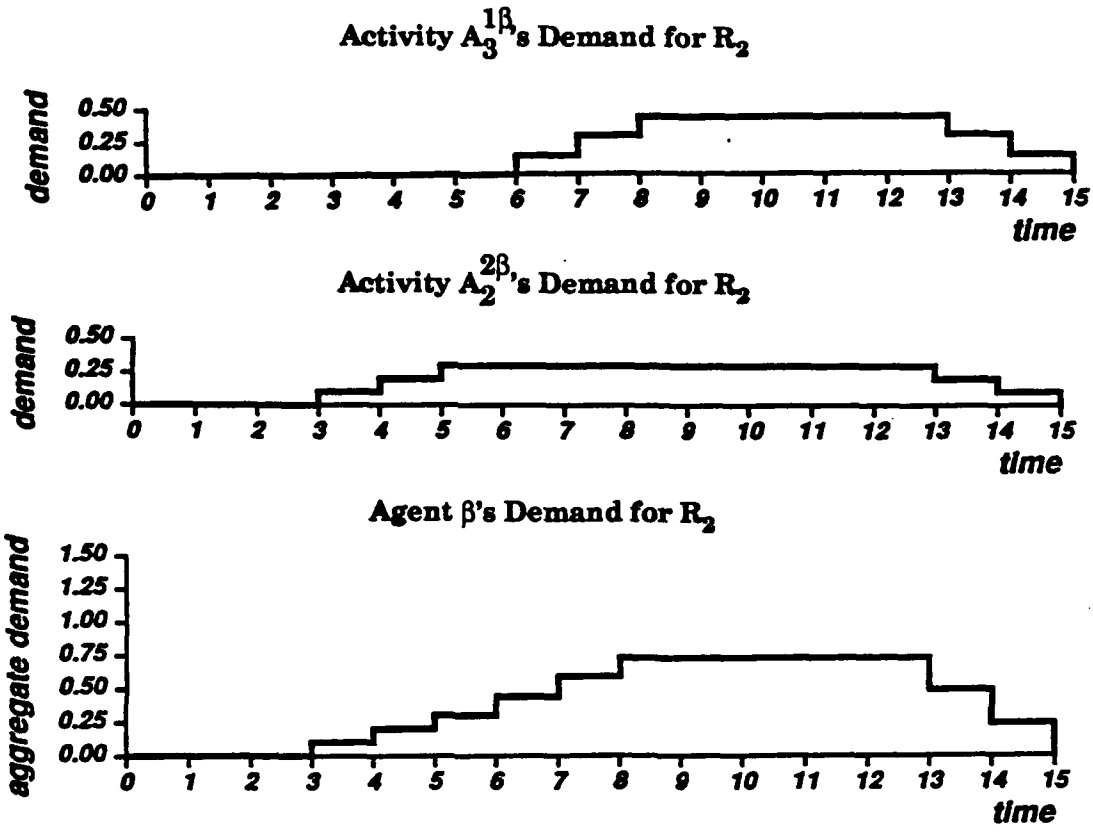


Figure 4-3: Building agent β's demand for resource R_2.

1/7=2/7, for interval [5, 10], it is 3/7, for interval [10, 11] it is 2/7 and for interval [11, 12] 1/7. The demand of activity $A_2^{2\alpha}$ is calculated in a similar fashion.

The two demands are then added up by the agent, thereby producing agent α's total demand for R_2. Notice that $A_2^{1\alpha}$ and $A_2^{2\alpha}$ have the same total demand. This total demand is equal to their duration, but it has been spread over the different possible start times of each activity. Because $A_2^{1\alpha}$ has fewer possible start times than $A_2^{2\alpha}$, its demand is more compact than that of $A_2^{2\alpha}$.

Finally, for each shared resource, agent demands are aggregated for the whole system thereby producing *system-aggregate* demands that indicate the degree of contention among agents for each of the (shared) resources in the system as a function of time. Time intervals over which a resource's system aggregate demand is very high correspond to violations of capacity constraints that are likely to go undetected by the constraint propagation mechanism. The contribution of an activity's demand to the system's aggregate demand for a resource over a highly contended-for time interval reflects the reliance of the activity on the possession of that resource/ time interval. It is taken to be the *criticality of the activity*.

In our example, agent α and agent β communicate their agent demands for each of the

shared resources to each other and aggregate them (i.e. take their sum) thereby obtaining the system's global demand for each resource as a function of time. So each agent knows the global systemwide demand at this point in time[28]. The exact communication protocol used by the scheduling agents is described in section 5. Figure 4-4 illustrates the aggregation process for resource R_2.[29]. At the end of this phase, agent α has three aggregate demand curves: one for each of the 3 shared resources, and agent β has 4 aggregate demand curves: the ones for the 3 shared resources and the one for its local resource R_4. These curves are represented in Figure 4-5. Notice that resource R_2 has the largest peak of demand, thereby indicating that it is the resource for which there is the highest contention (i.e. the main bottleneck resource).

To choose the next activity to schedule, each agent first looks at all the resource/time intervals on which it has some non-zero demand and picks the one with the highest aggregate demand. In our example, both agents α and β happen to have their highest aggregate demand on resource R_2. Therefore they both select time interval [8,11] as being their most contended one[30].

A second step to picking the activity to schedule next is for each agent to pick its activity with the highest contribution (i.e. highest criticality) to the aggregate demand for that resource/time interval. A higher demand contribution of an activity means that the activity is more likely to be involved in a capacity constraint conflict. In the example, agent α picks activity $A_2^{1\alpha}$ as its most critical activity, since it is the one (among its two activities contending for R_2 that relies most on the possession of that time interval (Figure 4-6) (i.e. the contribution of $A_2^{1\alpha}$ to the demand peak is larger than that of $A_2^{2\alpha}$). Similarly agent β picks $A_3^{1\beta}$ as its most critical activity.

4.4.2. Value Ordering Scheduling Heuristic

Once an agent has selected the activity to schedule next, it must decide which reservation to assign to that activity. Here several strategies can be considered. One type of value ordering heuristics is a least constraining one. The extremely small number of feasible solutions to a scheduling problem compared to the total number of schedules (including

---

[28]To avoid the computational cost of (a) each agent exchanging its agent demand with every agent it shares resources with, and (b) replicating the systemwide aggregation computation in each agent, in the implemented system, a number of agents, called monitors, one for each resource, play the role of systemwide demand density aggregators.

[29]Notice that in the case of resource R_4, there is no need for communication: R_4 is a local resource of agent β.

[30]The agents only consider time intervals of duration equal to the average duration of the activities requiring the resource, 3 time units in this example. Two time intervals actually qualify as most contended: [7,10] and [8,11]. We just assume that the agents both pick [8,11].

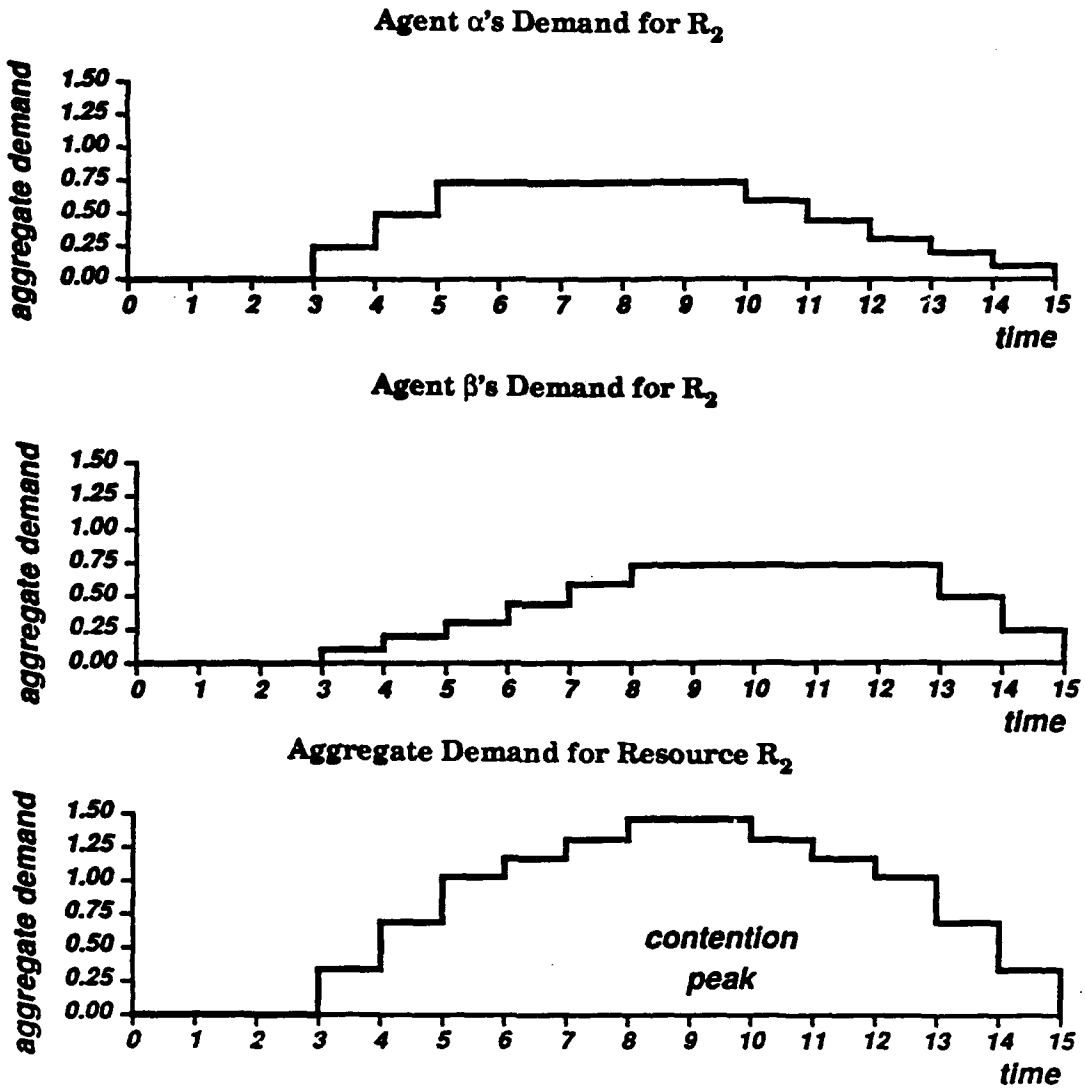


Figure 4-4: Agent and aggregate demands for resource R_2.

infeasible ones) that one can possibly generate is what has made least constraining value ordering heuristics so attractive. Agents using such heuristics attempt to select the reservation that is the least likely to cause constraint conflicts with reservations of other agents. In other words an agent will select the reservation that will be the least constraining both to itself and to other agents. This heuristic results in *altruistic* behavior on the part of the agent.[32] Because agents in the decentralized case schedule in an asynchronous fashion,

[32]There exist a variety of value ordering heuristics, such as the greedy Value Ordering Strategy (GV) where an agent can select reservations based solely on its local preferences, i.e. irrespective of its own future needs as well as those of other agents. In addition, there exist value ordering strategies that are intermediate between LCV and GV. These intermediate strategies attempt to factor in the contribution of a reservation to the global objective function together with the likelihood that selecting that reservation will result in backtracking (either locally or for another agent). The ultimate choice of an (intermediate) strategy is likely to depend on such factors as the time available to come up with a solution, the load of the agents, and the amount of resource contention. Experiments in centralized scheduling [Sadeh & Fox 89, Sadeh & Fox 90a, Sadeh 91] indicate that LCV-type heuristics are best at minimizing search, but usually result in poor schedules since reservations are selected irrespective of their contribution to the objective function. Value ordering heuristics of the greedy type usually produce significantly better schedules, but result in extra backtracking (i.e. search takes longer).

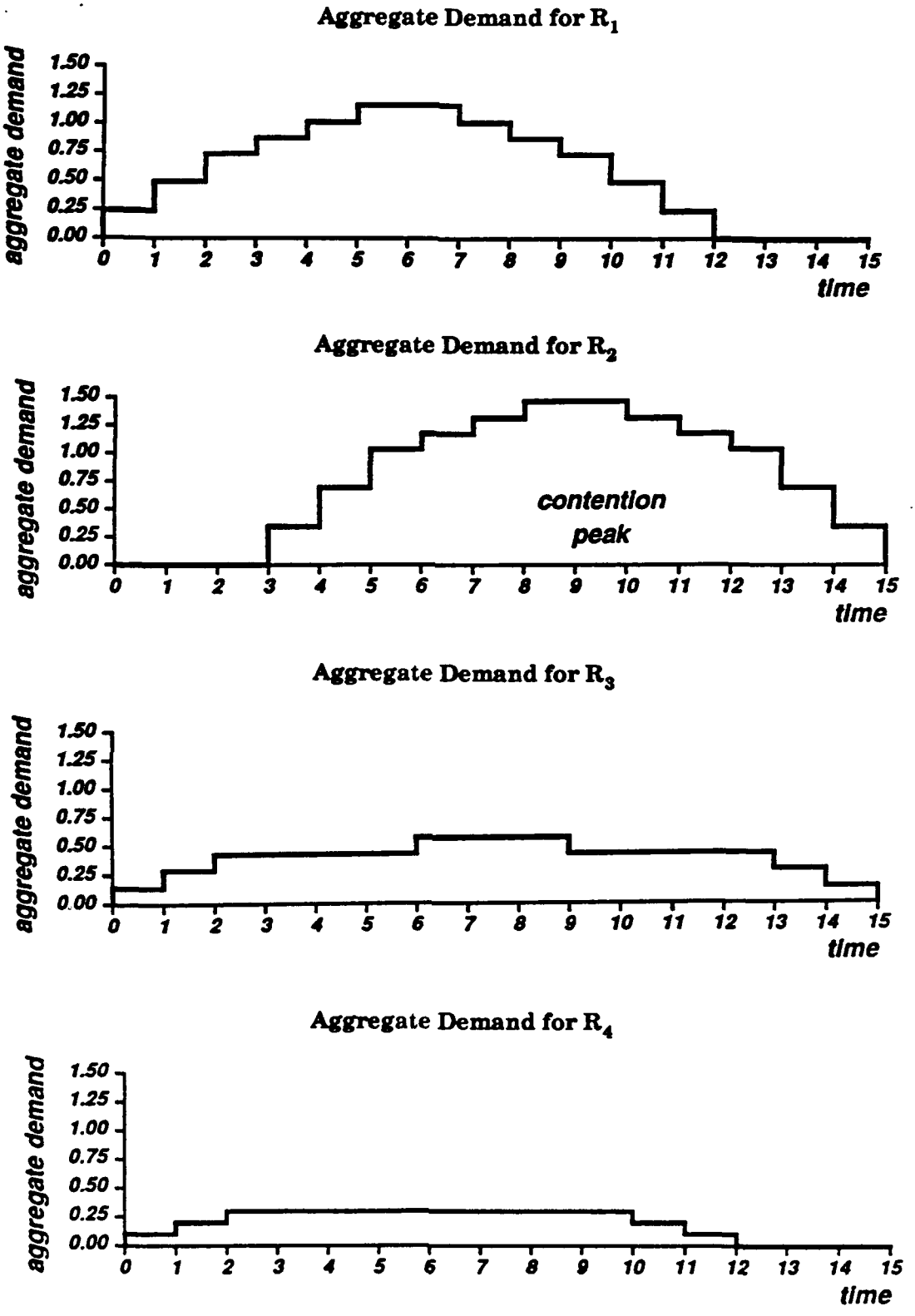


Figure 4-5: Aggregate demands for the three shared resource R_1, R_2, R_3 and the local resource R_4.

and because of the high cost of backtracking in such distributed systems, we expect a higher

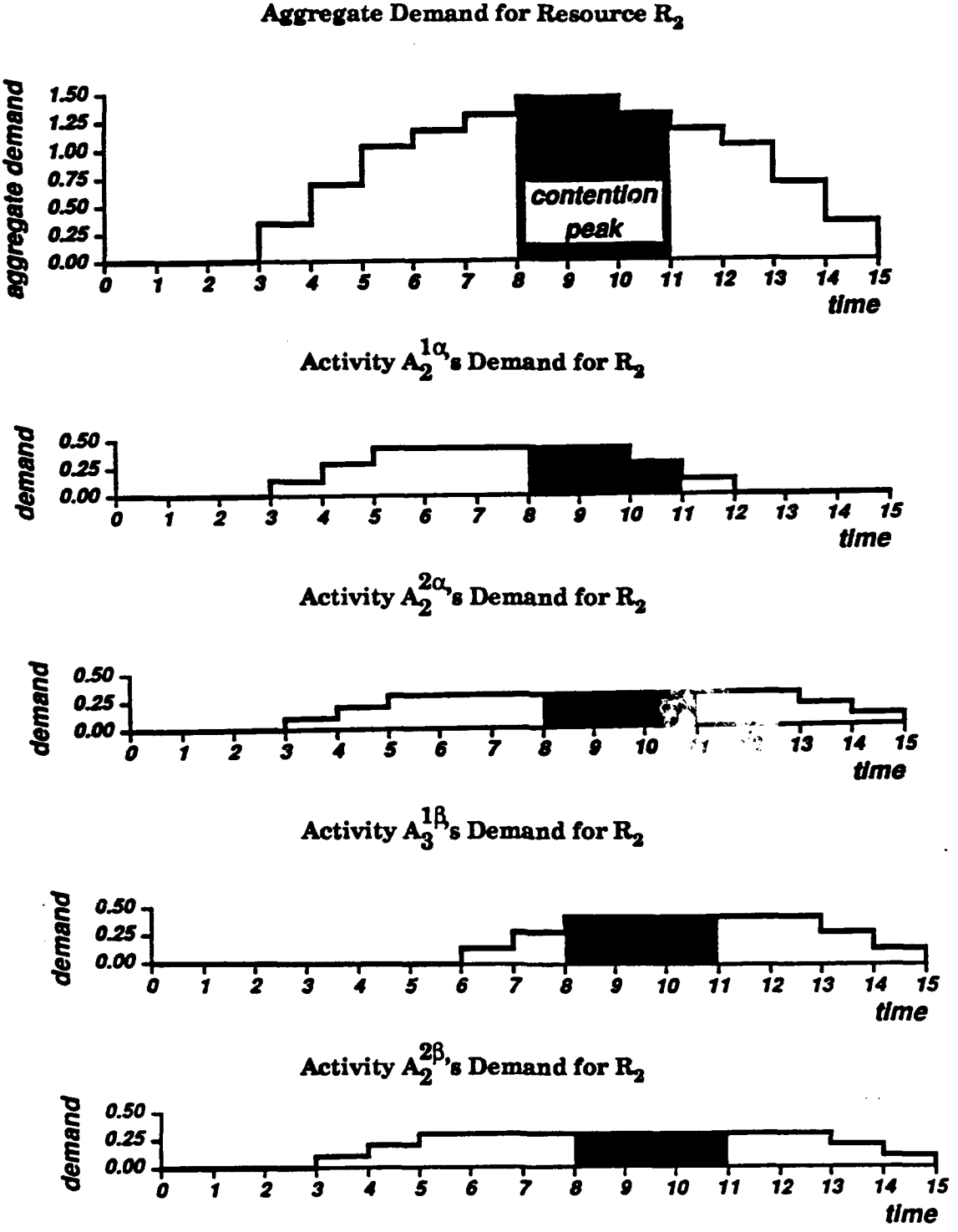


Figure 4-6: Activity ordering by agent α and β.

need for least constraining behavior in a distributed scheduling environment. LCV is a least constraining value ordering heuristic where every reservation for an activity A_k, is rated according to the probability that it would not conflict with another activity's reservation, if one were to first schedule all the other remaining activities. This probability is

approximated in our model by the ratio $\frac{D_{kij}(\tau)}{D^{aggr}_{R_{kij}}(\tau)}$, where $D_{kij}(\tau)$ is the selected activity's individual demand on each remaining available time interval (equal to the activity's duration), and $D^{aggr}_{R_{kij}}(\tau)$, is the systemwide aggregate demand for that time interval[32]. The reservation with the largest such probability is interpreted as the least constraining one and is selected for making a reservation.

Let us illustrate the calculation of the LCV heuristic in our two-agent example. As has been mentioned before, using the variable ordering heuristic, agent α has selected activity $A_2^{1\alpha}$ and agent β has selected activity $A_3^{1\beta}$. Now each agent will use the LCV heuristic calculation to determine the start time for its activity on resource R_2. The possible start times for activity $A_2^{1\alpha}$ are 3, 4, 5, 6, 7, 8, 9. The probabilities that a reservation made for each of these start times results in capacity conflicts for each of these start times are correspondingly 0.44, 0.42, 0.37, 0.33, 0.29, 0.28, and 0.26. We illustrate the calculation for start time 3: The activity demand over the interval [3, 6) (since the duration of the activity is 3 interval units) is given by 1/7+2/7+3/7=0.85. The system aggregate demand over the same interval is given by 0.30+0.60+1.00=1.90 (the values can be read directly from the two upper graphs of figure 4-6). The ratio 0.85/1.90=0.44. Since 0.44 is the largest of the probabilities thusly calculated, 3 is picked by agent α as the best start time for activity $A_2^{1\alpha}$. on resource R_2. Similarly, since the probabilities for the possible start times 6, 7, 8, 9, 10, 11, 12 of activity $A_3^{1\beta}$ of agent g[b] are 0.21, 0.29, 0.33, 0.35, 0.40, 0.44, and 0.46, agent β selects time 12 as the least constraining start time for $A_3^{1\beta}$. The least constraining value ordering heuristic works as expected: the agents end up selecting the non-overlapping intervals [3, 6) and [12, 15). Notice that all this is done without agent α knowing about the 2 activities of agent β, and without β knowing about those of agent α: the agents only exchanged their total agent demands, not their individual activity demands. In the LCV calculations each agent uses only its own selected activity's demand and the systemwide aggregate activity and no other knowledge.

4.4.3. Distributed Asynchronous Backjumping

As was discussed earlier, search is directed by variable and value ordering heuristics which select a resource, activity and time interval to schedule at each state. Assuming no other agent has reserved the resource interval, a reservation is made.

Although a resource has been allocated successfully to one activity at each state, it is still

---

[32]See [Sadeh & Fox 89] for detail. A more sophisticated least constraining value ordering heuristic that has allowed for even better performance is described in [Sadeh 91].

possible that a scheduling decision will prevent one or more still unscheduled activities from being scheduled within their start- and due-date constraints, either directly or indirectly. For example, a scheduling decision can directly prevent another activity from being scheduled if it reserves a resource during the only interval over which the unscheduled activity can use the resource. Relatedly, a decision can indirectly prevent another activity from being scheduled because of capacity and precedence constraints among activities. For example, the time interval for which an activity is scheduled can result in narrowing the range of remaining time intervals during which activities following the scheduled activity in a process plan can be scheduled. If the remaining time intervals for these latter activities are ones during which their required resources are unavailable, these latter activities cannot be scheduled.

Although temporal propagation is not able to detect all the indirect effects of a scheduling decision at each state, it is possible to detect t cases in which a scheduling decision makes it impossible for at least one remaining activity to be scheduled. We call this process feasibility checking and it is performed at every state, after a reservation has been made. Feasibility checking consists of: (a) time bound propagation based on activity precedence, and (b) determining whether each activity has at least one interval during which its required resource is available.

If feasibility checking is successful, the scheduling process continues to the next decision state. If it fails (for example assume the feasibility checking failed for start-time 20 for activity C in figure 4-7), in the standard centralized CSP chronological backtracking would occur[33]. Chronological backtracking involves the undoing of the last decision made by the system and substituting a different one. In a centralized system, this usually involves considering a different time interval for reserving a resource for the activity scheduled in the last state (activity C in figure 4-7). In the interests of readability, the figure depicts the various possible time intervals for scheduling an activity, just by the start time. When all remaining available time intervals (50, and 80) have been attempted for scheduling C and have failed feasibility checking, the backtracking process will undo the reservation (start-time 10) for the activity and resource scheduled in the **previous** state (activity B in figure 4-7) and attempt an alternative interval (start-time 30) for the previous state (state B). If activity B can be successfully scheduled at another time interval, search will "return" to attempt every possible time interval (including retrying the previously unsuccessful start-time 20) for the now unscheduled activity (activity C). If activity B cannot be successfully scheduled at any other time interval, then the process considers an alternative interval for activity A, then "returns" to consider reservations for the now unscheduled activity B and so

[33]In the figure the possible start-times for an activity, instead of being associated with new states, are denoted by lists appended to a state for better readability.

on. As a result, this process of making a reservation and feasibility checking is incremental and simultaneously tests the effects of all previous reservations made by an agent on activities remaining to be scheduled.

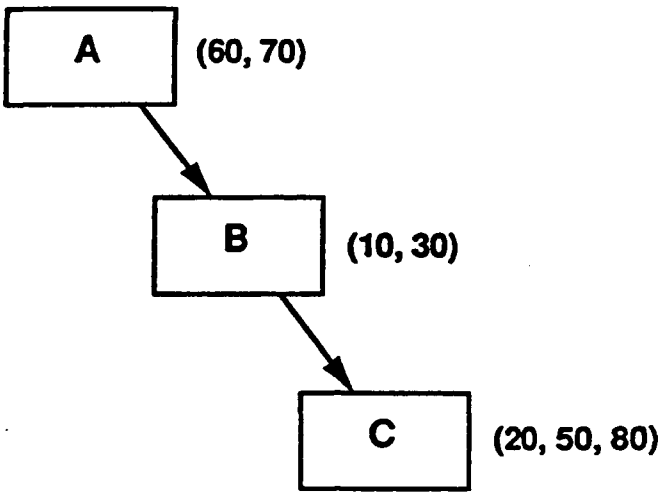


Figure 4-7: Partial State Space Search

In the multi-agent system, other agents can make reservations throughout an agent's search in an asynchronous manner, making it difficult for the agent to determine which set of previous reservations were responsible for a constraint violation when it is eventually detected. The task facing the agent at this point (when a violation is detected) is to find the last set of its own reservations which, together with those made by other agents, does not violate constraints. In the multi-agent case, suppose a reservation has been made by another agent since the last time our agent has performed feasibility checking. Then, after making a reservation, our agent performs feasibility checking and it fails. Chronological backtracking is no longer efficient because it **assumes that the combination of all previous activity reservations prior to the most recent reservation is still feasible.** In fact, it may no longer be the case that all previous reservations would pass the feasibility test, since other agents' reservations must be considered too. Therefore, the question is which subset of an agent's previous reservations, combined with the other agents' reservations will still produce a feasible state (one in which it is feasible that the remaining unscheduled activities can be scheduled).

Although chronological backtracking will eventually recognize this subset (if one exists) by trying every possible time interval for each previous activity-state and performing feasibility checking after each, the computational cost is enormous (exponential in the number of activities searched). To reduce the computational cost of backtracking, distributed asynchronous backjumping has been developed.

The backjumping process is as follows:

When infeasibility of a reservation is detected, prior to checking alternative reservations for this activity (and, if necessary for previously scheduled activities), an agent undoes the current infeasible reservation and tests to see whether the reservation of the immediately previously scheduled activity along with other agents' reservations remains feasible.

- If yes, then another reservation for the current activity is checked.
- If not, the process interleaves undoing of previous activity reservations with feasibility checking until a combination of previously made reservations of the agent along with other agents' reservations is feasible.

The cost of DAB is linear with the number of states (i.e. activities) scheduled.

To compare, consider again backtracking in figure 4-7: notice that before undoing activity C, every remaining time interval is attempted for activity B. Similarly, before activity A's reservation is tested for feasibility, every time interval of B must be tried (in combination with every time interval of C). In contrast, consider the backjumping procedure in figure 4-7. If the current reservation of activity C (start-time 20) is deemed infeasible (assuming other agents have made reservations since the last time feasibility testing was performed), the reservation is undone and, instead of trying alternative time intervals for C, backjumping performs feasibility testing on the remaining (excluding considerations of reservations for C) set of already made reservations (i.e., B and A's current reservations at start-times 10 and 60, along with the reservations made by other agents). If this feasibility check fails, then B's reservation at start-time 10 is undone and the process is repeated (i.e. now only A's reservation at start-time 60 along with the reservations made by other agents are considered). Finally, if A's reservation is feasible, search will resume with a different time interval on B (atart-time 30). In this case, backjumping has identified the first time interval tried for activity B as the one which produced the infeasible state. This was accomplished in just three tests. In contrast, it takes backtracking eight test to identify this situation. If A's reservation is infeasible, then an alternate time interval is considered for A and the process resumes. If all of the agent's reservations were undone and still feasibility checking failed, it means that the reservations made by other agents have forced the agent to an infeasible scheduling situation[34].

[34]This has been observed in some of our experiments where a particular decomposition of orders among agents results in a subset of the agents finishing with no backjumping whereas the remaining ones cannot find a solution.

4.5. The Communication Protocol

In the decentralized case, we have a set of agents that communicate in an asynchronous manner via message passing and each of which has a set of orders to schedule on a set of resources. Each order consists of several activities. Typically some of the resources are required by several agents and conversely, each agent requires some resources that are also needed by others. Which particular resources are shared may change with the set of orders to be scheduled. In our model, resources are passive objects that are *monitored* by active agents. Monitoring resources does not give an agent any preferential treatment concerning the allocation of the monitored resources but is simply a mechanism that enables the system to perform load balancing and efficient detection of capacity constraint violations. A capacity constraint violation (resource conflict) is detected when an agent requests a resource reservation for an activity for a time interval that is already reserved for another activity. Monitoring agents perform the additional tasks of (a) integrating certain pieces of information for shared resources (see step IV of protocol below) so as to avoid duplication of effort, which would be the case if all agents were doing this information integration, and (b) keeping the calendar of the resources they monitor. Typically, each agent in the system is a monitoring agent for some shared resources and conversely each resource is monitored by some agent. Since there is no single agent that has a global system view, the allocation of the shared resources must be done by collaboration of the agents that require these resources (the monitoring agent is usually one of those that require the shared resources)[35].

We have identified two levels of interaction of the agents: the strategic level where aggregate information is communicated and the tactical level where information about specific entities is communicated. The information communicated at the strategic level is the demand profiles out of which the agents calculate criticality measures for their decision making. At the tactical level, particular scheduling decisions are made and, if needed, negotiation takes place.

Because they may contend for the same resources, it is important that the scheduling agents build their schedules in a cooperative manner. The two texture measures identified in the previous section provide a framework for cooperation where the agents exchange demand profiles, and reservations. Demand profiles are aggregated periodically to compute textures that allow agents to form expectations about the resource demands of other agents. Because of communication overhead, the demand profile information is restricted. Subsets of the agents communicate only demand profiles for the resources that they share, although reservations on the non-shared resources may impact scheduling decisions on the shared

[35]This model mirrors actual factory floor situations where the factory is divided into work areas that might share resources, such as machines, fixtures and operators in order to process orders.

ones. Since several agents are scheduling asynchronously, and the communicated demand profiles are only those of the subset of shared resources, there is higher uncertainty in the system. This uncertainty also varies in an inversely proportional manner with the frequency at which the demand profiles are communicated. Moreover, the cost of backtracking is greater, since if an agent backtracks, the change in scheduling reservations may ripple through to the other agents and cause them to change their reservations.

In particular, the multi-agent communication protocol is as follows:

I. Each agent determines required resources by checking the process plans for the orders it has to schedule. It sends a message to each monitoring agent (as specified in a table of monitoring agent) informing it that it will be using shared resources.

II. Each agent calculates its demand profile for the resources (local and shared) that it needs.

III. Each agent determines whether its new demand profiles differ significantly from the ones it sent previously for shared resources. If its demand has changed, an agent will send it to the monitoring agent.

IV. The *monitoring agent combines* all *agent demands* when they are received and communicates the *aggregate demand* to all agents which share the resource[36].

V. Each agent uses the most recent aggregate demand it has received to find its most critical resource/time-interval pair and its most critical activity (the one with the greatest demand on this resource for this time interval). Since agents in general need to use a resource for different time intervals, the most critical activity and time interval for a resource will in general be different for different agents. The agent communicates this reservation request to the resource's monitoring agent and awaits a response.

VI. The monitoring agent, upon receiving these reservation requests, checks the resource calendar for resource availability. There are two cases:

1. If the resource is available for the requested time interval, the monitoring agent (a) communicates "Reservation OK" to the requesting agent, (b) marks the reservation on the resource calendar, and (c) communicates the reservation to all concerned agents (i.e. the agents that had sent positive demands on the resource).
2. If the resource had already been reserved for the requested interval, the request is denied. The agent whose request was denied will then attempt to substitute another reservation, if any others are feasible, or otherwise perform backjumping.

VII. Upon receipt of a message indicating its request was granted, an agent will perform consistency checking to determine whether any constraint violations have occurred. If none are detected, the agent proceeds to step II. Otherwise, backjumping occurs with undoing of reservations until a search state is reached which does not cause constraint violations. Any reservations which were undone during this phase are communicated to the monitor for distribution to other agents. After a consistent state is reached, the agent proceeds to step II.

The system terminates when all activities of all agents have been scheduled Backtracking, with this version of the protocol, is based on the following design decisions: 1) Once an agent has been granted a reservation, this reservation is not automatically undone when some

[36]With the exception of the first time demands are exchanged, agents do not wait for aggregate demands to be computed and returned prior to continuing their scheduling operations (although they can postpone further scheduling if desired).

other agent who had to backtrack now needs the reservation. This can lead to situations where one agent solves its local scheduling problem but the other agent cannot due to unresolvable constraint violations. 2) If an agent backtracks, it frees up resources but the reservation of other agents on these resources remain as they were. This policy may result in non-optimal reservation for other agents since it denies the other agents greater opportunity to take advantage of the canceled reservations of the backtracking agent, but it results in less computationally intensive performance.

4.6. Experiments with the multi-agent scheduling system

The goals of our experiments were to determine the feasibility of the texture approach to multi-agent scheduling, as well as to test particular mechanisms and parameters that influence system performance. In particular, our experiments considered:

- the effects of agents' incomplete knowledge of each other's plans (i.e. the robustness of texture measures when aggregated across multiple agents and with the resulting loss of detailed information),
- the effects of rapidly changing expectations on performance (i.e. the robustness of these measures with respect to delays in the communication of densities),
- the consequences of asynchronous scheduling (e.g., asynchronous use of variable-ordering strategies) without external coordination.

One of the goals of the experiments was to compare performance of multi-agent and centralized schedulers. The experiments summarized here were created from problems found to be difficult in previous research on centralized scheduling [Sadeh & Fox 89] and they reflect system performance with respect to search efficiency rather than schedule optimality. The problems were also selected and distributed across the agents in a way that maximized *resource coupling* within orders and across agents.

All the experiments were repeated with 1, 2, 3, and 4 agents. All experimental problems were selected so that orders could be distributed approximately evenly between the agents, all resources were shared by the agents (high inter-agent resource coupling), every order used all resources, and problems ranged from 40 - 100 activities. Over 150 experiments were run in order to vary several properties of each problem. In each case, the dependent variable was the efficiency with which the scheduling system found a solution. This was expressed by the total number of states needed to reach a solution. For example, for a problem with 40 activities, the minimum number of states needed to assign a reservation to each activity is 40. Every reservation that needed to be redone added an additional state to the total. This allowed comparing, for example, a 40-activity 1-agent problem to a pair of 20-activity problems solved simultaneously by 2 agents.

Problem versions differed in several ways. First, to establish a baseline, we created a 1-

agent system, which was similar to the multi-agent system in every way, except that the aggregate densities were constructed from a single agent. This was still different from the original centralized system in that decisions were based on an abstract aggregate (e.g. the aggregate did not include detailed information about the number of activities which contributed to the densities). Furthermore, it was possible to vary the frequency with which the aggregate was computed, thereby isolating the effect of uncertain expectations caused by infrequent and delayed communication of densities in the multi-agent system.

The timing of agent communication of their changed densities was determined by the following heuristic: in the MINIMUM delay condition, a single reservation on any resource by any agent initiated the exchange of densities for all resources; in the INCREASED delay conditions, densities were exchanged for each resource independently, whenever N reservations were made on it, where N = 1, 3, and 5. This provided a way to observe the effects of wide ranges in communication frequency (and hence the effect of information obsolescence) in the system.

Another version of the 1-agent system was created which used a *semi-random* (see Figure 6-1) version of the variable-ordering heuristic. The goal was to isolate and assess the effects of less accurate variable ordering that might occur in a multi-agent system. Recall that variable ordering is performed asynchronously in parallel in a multi-agent system (each agent selects the best activity to schedule from its subset of all activities which require a critical resource). Agents do not coordinate the selection of activities to schedule to ensure that the globally most critical ones are scheduled first. As a result, variable ordering is probably less effective than in a 1-agent system. The semi-random heuristic still selects activities to schedule from those which require the most critical resource/time-interval (which narrows the selection to a maximum of 20% of the activities in these problems). However, it then randomly selects from this subset, instead of selecting the activity with the greatest demand for the critical resource. Finally, two scheduler versions were created to compare the use of backtracking and backjumping search techniques.

The experimental results are presented in Figures 6-1 and 6-2 for representative groups of 40-activity and 100-activity experiments respectively.

The first important observation is that the use of texture measures was sufficient to allow near perfect performance when the texture information was updated frequently (MINIMUM Delay conditions). Thus, despite the incompleteness of information available in the various versions (different number of agents) of the multi-agent system, texture measures provide satisfactory summarizations. Second, as expected, performance of the multi-agent system does deteriorate as the communication of changing texture information is delayed. Since current texture information is used to perform both variable and value ordering, it is likely that both these processes deteriorate.

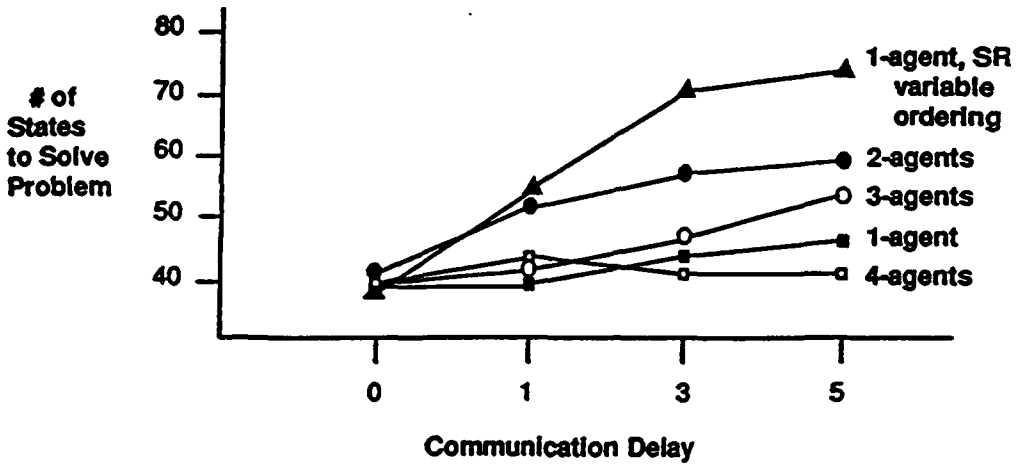


Figure 4-8: Experimental Results of 40-activity experiments

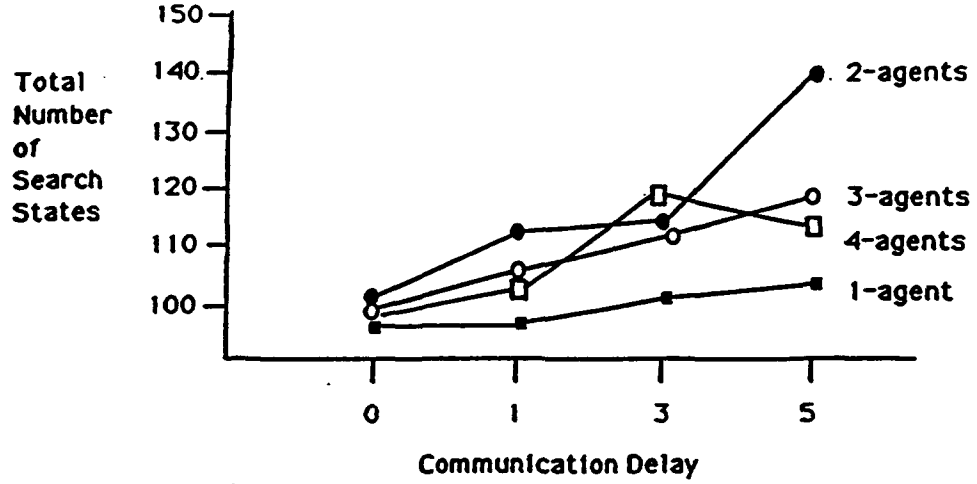


Figure 4-9: Experimental Results of 100-activity experiments

The effect of delaying communication/computation of demand densities is greater for all versions of the multi-agent than the 1-agent system. This interaction may reflect the compensatory relation between variable and value ordering [Sadeh & Fox 89]. In the multi-agent case, variable ordering may not be as effective because variables are chosen asynchronously. When texture measures are renewed frequently, there is valid information available to compensate for the poorer variable ordering by selecting effective values for variables. However, when density communication is delayed, value ordering is also weakened and the performance declines. In the 1-agent system, variable-ordering is more effective, so the delay does not hurt performance as much. This view is supported by the results in the 1-agent semi-random variable-ordering condition, where the effects of partially disabling variable ordering accelerates with increasing delay, as it does in the multi-agent case. Note that multi-agent performance is still better than the semi-random condition in all cases, suggesting that variable ordering strategy is robust with respect to the conditions of the multi-agent environment (incomplete, changeable information and asynchronous behavior without external coordination).

We should also note that the semi-random condition is still highly selective relative to completely random variable ordering (in that only activities which use the most critical resource/time interval are considered). In fact, we found that random variable ordering resulted in terrible performance, even in the 1-agent case. Solutions were not found in over 500 states. As expected, the use of a backjumping strategy substantially reduced the search in the multi-agent versions of the system. Performance using a chronological backtracking strategy was highly variable and degraded exponentially with delayed communication (searches exceeded 300 states when communication was delayed to after three reservations).

Another interesting observation from our experimentation was that different distributions of orders to agents produced different results in that one decompositions might result in feasible solutions by all agents whereas another might result in some of the agents finishing the scheduling of their assigned orders whereas the rest did not finish. Although we have not systematically performed experimentation with characteristics of the various decompositions, we hypothesize that agents with "easier" assignments manage to obtain the good reservations for their activities, causing solution infeasibilities for the rest of the agents. Studying the effect of different decompositions and their characteristics is one of the subjects of future research.

4.7. Conclusion

In this chapter we presented the Distributed Constraint Heuristic Search model and presented mechanisms to guide concurrent, asynchronous distributed search. In particular, we have presented measures of characteristics of a search space, called textures, that are used to focus the attention of agents during search and allow them to make good decisions both in terms of quality of system solution and performance. These textures play four important roles in distributed search: (1) they focus the attention of an agent to globally critical decision points in its local search space, (2) they provide guidance in making a particular decision at a decision point, (3) they are good predictive measures of the impact of local decisions on system goals, and (4) they are used to model beliefs and intentions of other agents. We have presented two types of textures, their operationalization into variable and value ordering heuristics and their use in distributed problem solving. A communication protocol that enables the agents to coordinate their decisions has been presented. We have developed a variant of dependency-directed backtracking, asynchronous backjumping, that substantially reduces backtracking costs.

Our investigation is conducted in the domain of distributed job-shop scheduling. A model of distributed job shop scheduling as an instance of DCHS has been developed and a testbed has been implemented that allows for experimentation with a variety of distributed protocols that use variable and value ordering heuristics based on the probabilistic framework

described in subsections 4.1 and 4.2. The experiments are designed to give results concerning the role of the heuristics in achieving search efficiency in distributed asynchronous DCHS under conditions of incomplete and rapidly changing information. The testbed is implemented in a decentralized manner in KnowledgeCraft running on top of Common Lisp, and can be run on a set of MICROVAX's 3200.

Future work will concentrate on developing strategies for adaptive use of various value ordering heuristics depending on where each agent is in its search when it has to make a reservation. In addition, we plan to develop negotiation protocols that will enable agents that contend for the same resource/time-interval (i.e. reservation) to negotiate over who should actually be granted the reservation. Criteria for making such a decision will include the relative priorities of the orders that the two contending agents have to schedule, how far each agent is in its scheduling, and whether an agent has access to a substitutable resource, or relies solely on using the contended-for resource.

Chapter 5
Constraint-Directed Planning

Summary

Planning is an important problem that has proved very difficult to automate. Much of the difficulty arises from the computational expense of searching for a plan within a sufficiently powerful representation. We present the CORTES approach to planning in CARMEMCO [Frederking & Chase 90], a model of a Computer Integrated Manufacturing (CIM) enterprise, which uses constraints to communicate the current concerns of plan users to the planner and constrain the search for an acceptable plan.

5.1. Introduction

There is a long history of planning research in artificial intelligence (AI), going back at least as far as the STRIPS [Fikes & Nilsson 71] system at SRI. In the development of planning, a number of properties have become widely recognized as important, such as domain independence, hierarchical planning [Sacerdoti 74], and non-linear planning [Sacerdoti 77]. Since we intend our planner to achieve all of these objectives, we will describe each of them briefly.

5.1.1. Classical planning

For a planner to be interesting from an AI point of view, it must be domain independent. If a planner is not domain independent, it can be very difficult to tell what is happening in particular, whether it succeeds in planning because of its planning capabilities or because of clever, domain-specific representational and design decisions by its human creators. Domain independence helps guarantee that it is the planning representations and operations that are doing the work.

Hierarchical planning is necessary if a planner is to take on real-world scale problems. Solving a large problem from beginning to end at the finest level of detail is completely intractable, even with heuristic AI methods. This is remedied by first solving the problem at a very abstract level, and then refining each of the abstract activities into a set of activities

at a finer level of detail, repeating until a ground level is reached in which the activities are primitive domain actions.

When a planner is asked to satisfy a conjunction of goals, the problem of non-linear planning arises. Early planners simply planned each conjunct in turn, and strung the resulting plans together, "linearly". This produces badly non-optimal solutions, and can fail on problems that are actually soluble. The traditional solution to this is to produce conceptually parallel solutions for the conjuncts, represented as a partially ordered plan. Since in reality there usually are interactions between some of the goals, "critics" are used to add ordering constraints to these plans where necessary to make them correct. This approach to non-linearity is cumbersome and very expensive computationally. We will discuss the approach to non-linearity we use in section 3 below.

5.1.2. Replanning and constraints

Recently, the ability to replan in reaction to a changing world has been emphasized [Wilkins 88]. Since this implies that the planner is reacting during execution, efficiency is especially important. The planner must be able to recognize and retain those parts of an old plan that are still valid, replace the invalid parts with unexpanded nodes specifying the goals that they need to achieve, and generate a new plan from this point. One can view this process as dependency-directed backtracking [Prosser 89].

Since a planner in a CIM environment must interact with the users of its plans, replanning in reaction to changing user needs is central to our effort. The communication necessary for this interaction is achieved in our environment by having the user programs communicate constraint information to the planner. These constraints are used to guide the planner in its search for a good plan. The criteria for the "goodness" of a plan have traditionally been measures on its graph structure, such as having the fewest total non-phantom nodes. This does not make sense for us, since planning is not an end unto itself. Our plans do not exist in a vacuum; the users of our plans can provide us with realistic criteria for their quality. It will often be the case that a plan containing several inexpensive nodes is better than one with a single expensive node.

In order for the user programs to communicate their constraints to the planner, there must be some kind of language that supports such communication. This constraint language, representing a wide variety of constraints on knowledge base objects, must be understandable to both the planner and the plan user. This will allow the expression of a wide variety of criteria for plan quality to the planner interactively, instead of having those criteria built in to the planner's architecture.

5.1.3. Criticality and constraints

The constraints expressed in this language also play a key role in the solution of a major difficulty in planning. Because planning is an NP-complete problem, any planner that is to solve realistic planning problems must somehow find a way to reduce its searching to a manageable size. This has been done in a variety of ways, including the use of "criticality". This term has been used in a number of ways in the planning literature to describe means for deciding the order in which to insert operators into a plan. For example, in ABSTRIPS [Sacerdoti 74] the so-called "abstraction levels" are actually criticality levels for domain predicates. A plan is found involving only the most critical preconditions first. Then preconditions at the next level of criticality are considered, and so on, until all criticality levels have been considered. There are only a few levels of criticality, with many predicates on each level. Also, the criticalities of the predicates are static, being determined once for each domain.

Unlike earlier planning systems, we intend to make criticality decisions on an action-by-action basis, dynamically deciding during planning which goal, operator, or variable instantiation is the most critical, based on current constraint information. This information includes constraints communicated to the planner by the users of its plans, in addition to information from *a priori* constraint graphs, also represented in terms of our constraint language[37]. Some of these *a priori* constraints may be automatically derived from the domain description. This use of constraints to guide the planner in its search for a plan is what we mean by constraint-*directed* planning.

Some motivating examples will make the need for constraint-directed planning clear. Since we will be producing production and process plans, the CORTES scheduler will be a major user of our plans. If the scheduler detects a bottleneck resource while attempting to schedule a set of orders, it could ask the planner to replan to avoid this resource. The preference to avoid the bottleneck resource, described in terms of a constraint, would then be a major factor in operator selection, possibly causing the selection of operators that would not have even been considered otherwise. As another example, consider an order whose due date has been suddenly advanced, causing pressure for immediate completion. A constraint-directed planner could provide the scheduler with a new process plan that meets the new due date, but with a higher cost than the original plan. As a final example, if the production of a particular order becomes extremely costly, the planner could rework the original plan to minimize cost, perhaps at the expense of delaying completion. This delay would be acceptable given the new set of constraints, even if it was not with the original constraints.

---

[37]The formulation of constraint graphs appropriate to planning is currently under investigation.

5.1.4. Related work

The use of criticality to opportunistically select the next decision to make is related to other work in planning, as well as to work in opportunistic scheduling.

Relevant previous work in planning concerns goal ordering and operator selection. STRIPS [Fikes & Nilsson 71], in addition to using means-ends analysis to select operators, had a search strategy based on subgoaling. When an operator was instantiated, STRIPS would next try to achieve any unachieved preconditions of that operator. If successful, the operator would be "applied", changing the state. If the new state did not match the goal state, a new means-ends analysis would be done. As discussed above, ABSTRIPS [Sacerdoti 74] in effect used criticality to order subgoals. SIPE [Wilkins 88] achieves some operator selection by programmer-specified conditions on its operators that determine applicability. Hayes-Roth and Hayes-Roth [Hayes-Roth et al. 79] have a blackboard based planning system with a notion of opportunism. If the system heuristically realizes that some operation will be necessary, it is added to the plan, with ordering determined later.

Our constraint-directed selection of planning goals, operators, and variable instantiations is inspired by the identification of critical scheduling decisions in opportunistic schedulers, such as OPIS [Smith & Ow 85] and CORTES [Fox & Sycara 90] [Sadeh & Fox 89]. When OPIS detects a bottleneck resource, it opportunistically switches from an order-centered to a resource-centered viewpoint, in an attempt to make the most critical (most constraining) decisions first. After the bottleneck resource is scheduled, it switches back to an order-centered viewpoint. CORTES takes this approach a step further, using "texture" measures (criticality measures) to micro-opportunistically select the next decision to make on an activity-by-activity basis. The concept of *a priori* constraints mentioned above is based on CORTES's use of textures.

5.2. Representing Plans: The Activity-State Network

In describing any manufacturing environment it is critical to capture the processes which can occur. The mechanism for describing processes in the CARMEMCO model [Frederking & Chase 90] is the activity object. An activity is a node which links resources, parts, production units, and raw materials together to represent how the interactions among these objects produce change. Activities are linked to pre- and post-state objects which describe what's true about the world before and after the activity executes. Activities can be described at multiple levels of detail; they are linked temporally to create process plans.

State objects provide a specific mechanism for describing and tracking the changes that activities make. Each activity has a pre-state which describes what must be true in order for

the activity to occur. Each activity also has a post-state which describes what will be true after the activity occurs. A simple state in CARMEMCO encapsulates a logical expression which refers to knowledge base objects and a time interval during which this logical expression must be (or is) true. For instance, an assembly activity might have a pre-state which requires that all of the activity's input production units be present at a particular machine at the same time before assembly can begin. Arbitrarily complex states which are disjuncts and conjuncts of simple states can be built and used as pre- and post-states for activities.

The time intervals used in descriptions of states can be actual time stamps or relative time relations between states. We exploit Allen's thirteen temporal relations[38] between time intervals in our state representation [Allen 83]. We derive the appropriate time interval for complex states' extent using Allen's time interval calculus.

The activity-state network representation for processes and system states was originally developed for use in the Callisto project management project [Sathi et al. 85]. The activity-state representation is currently used in several projects at CIMDS, including SAGE [Roth & Mattis 88], an intelligent human interface manager.

We will now give a detailed example of a process plan implemented using the activity-state network approach. The part we have selected for illustration is the articulated arm from the CARMEMCO model.

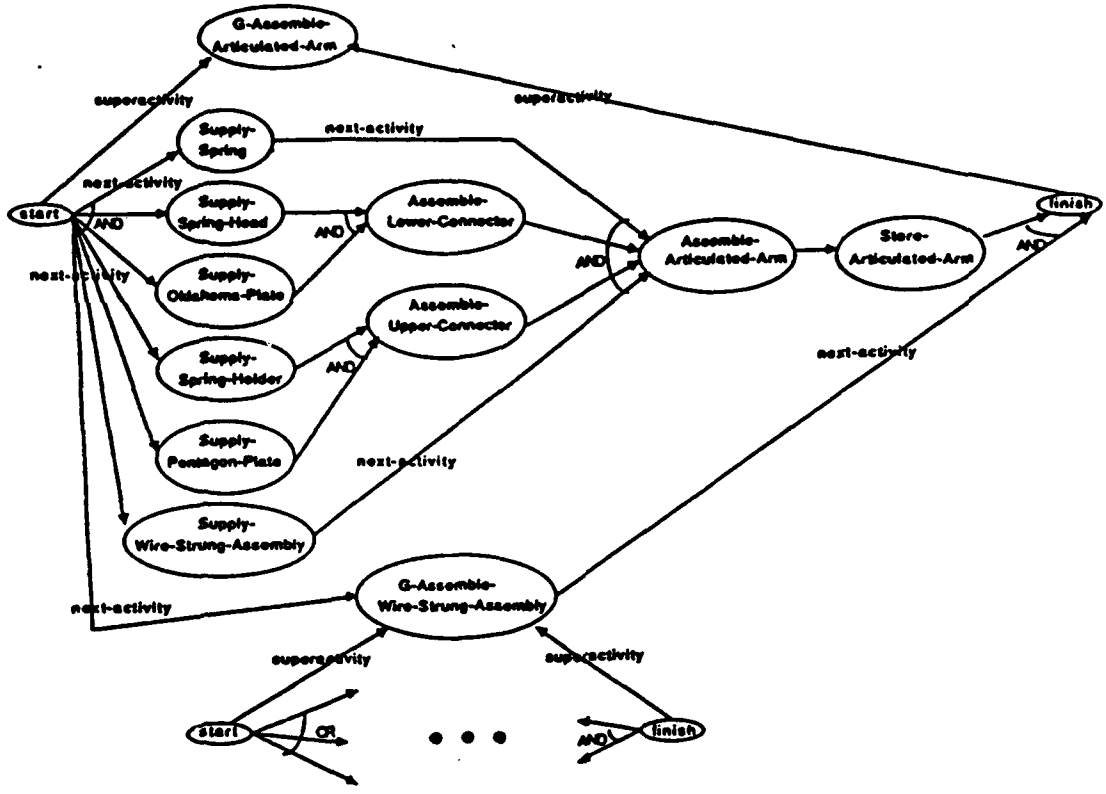
In Figure 5-1 we have the activity nodes which are linked to form the process plan for assembly of the articulated arm. This example illustrates two important topological characteristics of activity descriptions. The node g-assemble-articulated-arm represents the entire process plan at the highest level of abstraction. It is elaborated into one level of greater detail via the subactivity relation. This elaboration is recursive in nature, as is illustrated by the further elaboration of g-assemble-wire-strung-assembly. This ability to describe activities hierarchically is the first important feature of activity descriptions.

Second, note that within a layer of the activity network temporal ordering relations are expressed via the next-activity and previous-activity relations. Partial ordering of activities is provided with both disjunctive and conjunctive branching. (In this case only conjunction is illustrated.) This (temporal) partial ordering of activities is the second important topological characteristic of activity networks.

In Figure 5-2 we see a detailed example of the network links which are typical for each

[38]equal, before, after, during, contains, overlaps, overlapped-by, meets, met-by, starts, started-by, finishes, finished-by

Figure 5-1: Upper Levels of the Prototypical Process Plan for the Articulated Arm



activity in a network. Some links to the activity such as previous-activity, next-activity, and superactivity, are topological in nature. In this case we have an assembly activity which requires that two types of parts be supplied before assembly can begin. This assembly activity produces a subassembly which is fed downstream to another assembly activity, assemble-articulated-arm. Other links such as input-component and output-component connect the activity with the parts which the activity consumes and creates. The pre-state and post-state links connect the activity with state descriptors which check and assert conditions in the knowledge base before and after execution of the activity. In this case the pre-state is checking that the proper enabling activities have completed, that all necessary production units have been enqueued, and that one fixed resource of the appropriate type is

available. All of these conditions must be true at the same time for the state AUC-pre-state to evaluate to TRUE. The post-state is asserting that the assembly activity has been completed, that an output production unit of the correct type has been produced, and that the resource which was in use has been released. (Other types of references which typically occur in states are to availability of mobile resources and raw materials.)

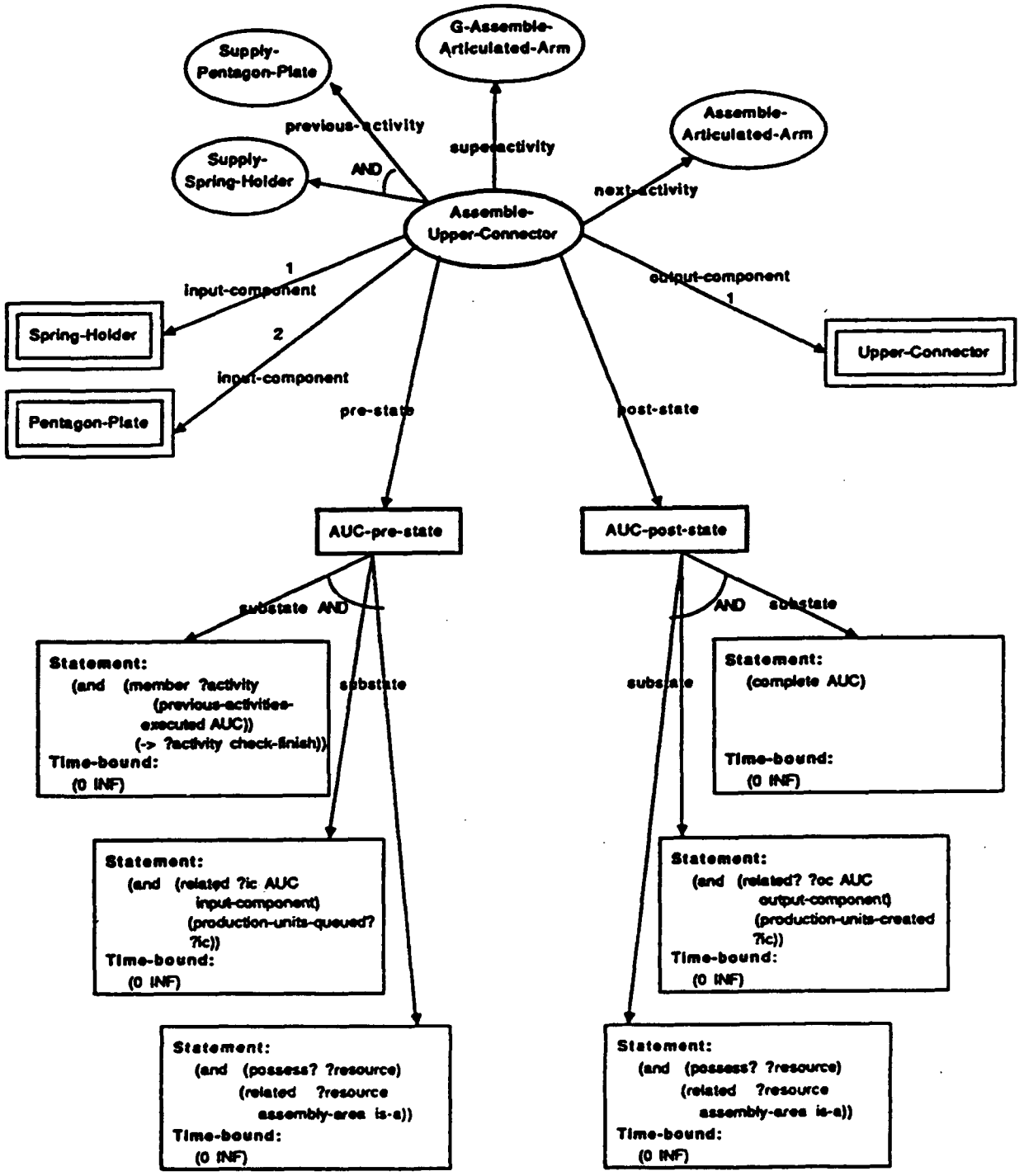
Several features of our activity-state networks are extensions beyond the original work by Sathi. Specifically, the ability to refer to knowledge base objects via variables in states is a new contribution. This relaxation of the previous requirement that references occur only to ground instances provides a significant increase in expressive power. For example, partial specification of plan operators is now possible. The integrated use of Allen's time algebra with the state representation is also our extension.

Since the activity-state network was already in use in our CIM environment, it was the obvious candidate for our plan representation language. Our extended activity-state network language has expressive power equal to the plan representations in several well-known AI planners such as STRIPS [Fikes & Nilsson 71] and SIPE [Wilkins 88]. In the degenerate case of single previous- and next-activity relations among activities the linear plans of STRIPS can be constructed. The pre- and post-state relations and state objects provide exactly the language needed to replicate the STRIPS operators' add and delete lists. Equivalence of activity-state representations to the SIPE operator description language is attained through the features already mentioned together with use of multiple previous- and next-activity relations and the hierarchical elaboration of activities via the superactivity and subactivity links.

5.3. Creating Plans: A Testbed Planner

We have begun our investigations into creating a constraint-directed, non-linear, hierarchical planner by building a simple non-linear, hierarchical planner. It has been designed for flexibility and clarity of operation, and is intended to serve as a testbed for developing criticality measures, flexible hierarchical planning capabilities, and dependency-directed replanning capabilities. This initial version the planner works on blocks-world problems, the Tower of Hanoi, and the planning of simulation experiments, using the mechanisms and data structures that underlie the CARMEMCO model.

Figure 5-2: Detail of the Assemble-Upper-Connector Activity



5.3.1. "Non-linear" planning

As mentioned in section 1 above, our planner incorporates a new approach to non-linearity, which is inspired by the work of Veloso [Veloso 89]. She points out that the deep problem with "linearity" is not the linearity of the plan representation used, but the "linearity" of completely solving one goal before working on the next. This was a deliberate heuristic used in the early planners, to reduce complexity. When "non-linearity" was introduced, it was in the guise of partially ordered plans. Unfortunately, the partially ordered representation makes the computation of the truth of even one predicate very expensive. From our viewpoint, the problem is that this approach delays some of the most critical decisions, which ought to be made first.

The other option, to retain the much simpler linear representation but allow free selection of the next goal to consider, was mentioned but not pursued. This approach does increase the size of the search space, since a possibly incorrect choice for a position must be made when adding a new step to the plan, and the planner must (at least potentially) consider all possible orderings of the steps. However, the search space is large enough in either approach that heuristics are necessary for non-trivial problems. The linear representation allows conceptually simpler and computationally cheaper basic operations[39], and consolidates all the heuristics in a single place: the search space.

5.3.2. Planning using activity-state networks[40]

As we have said, our plans are represented as activity-state networks. Operators are abstract activities with additional slots indicating their planning variables, preconditions, and postconditions. A planning problem is represented as an initial state and goal state connected by an empty high-level plan, representing that something as yet undetermined will produce the not-fully-specified goal state from the initial state. When the first operator is instantiated into a plan activity, its pre- and post-states are created from the initial and goal states, modified by the action the activity represents.

As steps are added to the plan, intermediate states are added between the steps as well, explicitly representing both the predicates known to be true at each point in the plan, and unsatisfied goals at that point. In the simplest application of the planner, each search step either adds a plan step to the plan or instantiates an uninstantiated variable. Mistaken choices lead to backtracking and the use of another alternative.

---

[39]It is a "vivid" representation for plans [Etherington et al. 89] [Levesque 86].

[40]This section differs significantly from the description in [Frederking & Chase 90], due to advances in the planner's design. In particular, "null" plans no longer exist.

The possible steps to add to a plan are determined by means-ends analysis (MEA) of the states in the plan. This produces a set of partially instantiated operators, each of which will achieve some goal that exists in a state with no matching predicate. In later steps the other variables in the operator are instantiated. Variable instantiation is also driven by an appropriate version of MEA. Taking either of these steps can add new predicates to a state.

Predicates added to a state are propagated forwards through any operators that do not affect them to the states on the other side, while goals are propagated backwards. Thus, a predicate exists from the point where it is created (by an operator or the initial state) up to the point where it is destroyed by another operator. Similarly, goals exist from the point at which they are created (by a precondition of an operator or the goal state) backwards to the point where they are destroyed (by the creation of a matching predicate). A difference exists in a state when there is a goal that has no corresponding predicate. This creates a complete picture of why things are happening in the plan.

This process is how we achieve "non-linearity", since the planner can select any open goal to work on next; they do not need to be subgoals of the same goal. There is no built-in enforcement of any search strategy through the currently unsatisfied goals.

In order to implement this form of MEA, it is necessary to do a simple form of dependency maintenance for individual predicates as they propagate through states. This allows the system to tell where a predicate came from, and thus discern the subgoaling structure of the plan. To implement this predicate source tracking, each predicate in each state is marked to indicate how it was introduced into the plan. When predicates are propagated to new states, they carry this information with them. When MEA is carried out, subgoals are created by comparing predicates without regard to their sources, but the subgoals thus created still indicate their sources. In other words, a subgoal indicates the need for a predicate without specifying where it should come from, but does indicate where the need for the predicate arose.

The subgoal structure is also necessary if one is to detect subgoal loops or multiply achieved goals, or imitate the STRIPS search strategy. The first two are currently done. When the goal of a step is a subgoal (to any depth) of a step already taken to achieve the same goal, the new step's line of search is terminated. (Otherwise it could loop indefinitely, without any actual progress.) In multiply-achieved-goal elimination, if the same goal from the same source that the current step would achieve has already been achieved earlier in this search path, this path is terminated. The motivation for this is that there must be a shorter plan that only achieves the goal once, and this path could lead to an infinite loop similar to those in subgoal looping. We believe this is related to the well-known technique of "goal protection", where a planner will refuse to introduce an operator that would destroy a predicate between where it is created and where it is needed. This technique achieves the same results by different means.

The MEA for variable instantiation mentioned above solves a common problem in planners. It looks for a variable instantiation that does *not* create a precondition that does *not* exist in the preceding state. For example, in the blocks-world, if the planner has the goal of clearing block C, it generates a partially-instantiated move from on top of block C, without specifying what to move or where to move it to (due to the way the operators are encoded, it doesn't know that it must move the block already on block C). Our variable MEA will produce the block on top of C as the only choice for the block to be moved, since any other choice would create a precondition that does not already exist. Many planners (including earlier versions of this one) would stupidly try to move blocks onto C to have them available for moving off of C! If no such "good" instantiation exists, the planner reverts to the usual blind instantiation of all possibilities.

There are currently four selectable search strategies: bounded depth-first search, breadth-first search, user-selection of the next step, and user-generation of the next step. User-selection of the next step is selection from steps suggested by MEA, while user-generation allows the user to select the next activity to add to the plan and its location without regard to MEA. Thus the full range of possible actions is made available. At any point in the search the planner can be switched from one strategy to another. User-selection and user-generation should be useful in developing heuristics and analyzing domain constraints, as well as being a useful capability in its own right in a man/machine planning system.

Because operators can be introduced in any order, it is possible for the same plan to be arrived at via different search paths. The inefficiency this could cause has been remedied by maintaining a plan hash table. Every partial plan produced is hashed into this table. When a new search step is taken, the resulting plan is checked against the hash table. If it is found, the step is abandoned, since this partial plan is already being investigated somewhere else in the search tree. This has saved over 60% of the search space in deep searches. If dependency-directed backtracking is introduced, dependency information for all paths leading to a search node will have to be stored via the hash table, so that when a path is invalidated it can be determined whether the search node is still valid due to other paths.

We have included the planning of factory simulation experiments as a domain. This domain is complex enough to demand hierarchical planning. We have implemented truly hierarchical planning, where higher-level activities are abstractions of groups of lower-level activities and do not necessarily appear in the final, ground-level plan. In hierarchical domains, high-level operators indicate a list of suboperators that should be used to expand them. When hierarchical expansion of a node is called for, the planner is recursively invoked on the selected node, using its preceding and following states as the initial state and goal, and its suboperators as the available operators. (Eventually there will also have to be means for mapping between high-level and low-level states.) Success at a lower level results in

marking the parent activity as successfully expanded, and, if automatic planning is in use, returning to the next higher level. Under manual (user-generated) planning, the lower levels can be re-entered, and the user can return to the higher level at any time. If lower-level activities are only partially instantiated, and the user returns to a higher level and then invokes automatic planning, it will return to the lower level and finish instantiating the activities there in the course of its planning.

In its simplest applications, the planner is not fully opportunistic. "Full opportunism" is the ability to select an operator for addition to the plan, and instantiate some of its parameters, without specifying where it should go in the plan. A relatively simple change to the planner allows full opportunism: the operation that inserts a new step has been divided into two parts, one that creates a step, and a second that inserts the step into its place in the plan. In the initial version of this, the user must eventually specify a total ordering before starting depth-first search, since the depth-first search strategy used assumes totally ordered plans. Eventually it should be possible to have the planner create all total ordering possibilities that satisfy the plan's goals as search alternatives. The initial version also allows only one ordered set of activities; that is, there is exactly one totally ordered partial plan and a set of unordered activities. Later versions should allow more than one totally ordered partial plan, and should eventually allow partially ordered partial plans as well. The propagation of predicates with dependencies described above becomes distinctly non-trivial in this context.

We have added static preconditions to our operators. This allows an operator to require that any of its instantiations satisfy certain predicates, without allowing the predicates to become new goals. Our encoding of the Tower of Hanoi makes use of this feature, since whether a disc can be set on another disc depends on its size, but no domain operator can alter a disc's size. Without static preconditions, useless subgoals would be produced to make one disc bigger than another. This feature works by causing the failure of a variable instantiation if any of the static preconditions are made false[41].

The planner includes an initial version of a graphical interface. Plans can be built interactively (using user-generation), automatically (using bounded depth-first search), or a mixture of the two. The user interactively plans by clicking on graphical buttons representing all the available operators. The instantiated operators in the current partial plan are shown in another window. Open variables in the instantiated operators can be bound by clicking on them in the plan: a menu then pops up, presenting the available choices. When finished specifying a partial plan, the user clicks on "Automatic Planning

[41]Instantiation failures can also be caused by predicate propagation, if a predicate propagated to a state clashes with a postcondition of the step before that state.

Search", which fills in the rest of the plan using MEA. As the planner alters the plan, the changes appear in the plan window. The set of available operators that appears changes to show those currently under consideration due to MEA. Much further work needs to be done before the interface has all the desirable functionality, but it is currently quite usable.

5.3.3. Future developments

The planner cannot currently build activity-state networks with parallel activities. This should be possible within our approach, with the planner making two sets of activities parallel sequences once it knows that they do not interact. This is a stronger form of non-interaction than is normally assumed by partially-ordered planners, since they usually assume that the plan will be linear when actually executed.

5.4. Conclusion

We have presented here the initial steps of a research program leading to the development of a constraint-directed planner designed to interact with other elements of a CIM environment. The constraint-directed nature of the planner will allow it to construct plans that correspond to the current needs of the users of its plans and to reduce its search space to a feasible size. Given the success of constraint-directed search in scheduling, we expect that the application of these techniques to planning will yield a powerful, domain independent planner that automatically adapts to changing plan requirements.

Chapter 6

Protection Against Uncertainty In a Deterministic Schedule

6.1. Introduction

We focus on the relationship between domain uncertainty and temporal precision in the development and execution of schedules. It is obvious that as uncertainty increases in a domain, it is less likely that a predictive schedule will be implemented successfully and it is more likely that there will be a greater reliance on a reactive scheduler to generate an appropriate response [Fox & Smith 84, Ow et. al. 88]. Consequently, spending more time on developing precise schedules that further optimize a set of measures, such as reducing work-in-process or tardiness, may be unnecessary if the temporal granularity of the impact of uncertainty exceeds the granularity of the optimization of the schedule. But given the necessity of developing predictive schedules, in order to plan resource purchases, allocations and releases, the question arises as to how precise should temporal decisions be, and the nature of the flexibility that a reactive scheduler should be afforded in responding to stochastic events. The domain in which we explore these issues is manufacturing scheduling.

It is a fact of manufacturing life that machines inevitably malfunction, materials fail, or resources are not available when required. Of the many effects of such an event, operational schedule disruptions are perhaps the most visible ramifications. A disrupted schedule incurs higher costs due to missed customer delivery dates, higher work-in-process inventory, and idling of people or machines. One may ask why should schedules be generated at all? Perhaps a reactive approach to scheduling should be taken as found in situated control or action networks [Schoppers 89, Ginsberg 89, Nilsson 89]. Predictive schedules are necessary to reduce costs so that resource requirements can be determined and their purchasing and allocation can be planned for. A critical decision in scheduling in the presense of uncertainty is *not* when an operation is to be scheduled, but when the materials are to be released to the factory. The earlier the materials are released the longer they will wait on the floor until the machine and personnel are available for the operation to be performed. On the other hand, the later the materials are released, the greater potential idleness of machines and people because the material is not available when the opportunity arises that the machines and

people are. Consequently, when to make resources available on the factory floor affects the amount of work-in-process inventory, the idle time of personnel and machines, and the ability to meet due dates. A second critical decision is to determine how long an operation is to be performed. If too little time is set aside, then subsequent scheduled operations will be incorrect. If too much is set aside, then subsequent operations will have too much embedded idle time. The release time and the amount of operation duration slack are defined to be an operation's ***temporal protection***.

The goal of our uncertainty management research is to dynamically construct a model of uncertainty in a manufacturing setting, by monitoring the occurrence of stochastic events, in order to determine the amount of temporal protection. As uncertainty increases, the model would increase the amount of temporal protection, thereby decreasing the precision of the predictive scheduler, while at the same time providing the reactive scheduler (aka dispatcher) with greater flexibility in its ability to react to change.

The experiments described in this chapter focus on the single machine scheduling problem with variable size jobs and stochastic arrival times. Three metrics used to evaluate adjusted schedules are (1) the resulting work-in-process levels, (2) the tardiness induced by the schedule, and (3) machine/personnel idleness.

This research is intended to be one of the modules in the distributed manufacturing project CORTES [Fox and Sycara (1990)]. This part of the CORTES system consists of the following modules: (1) Uncertainty Analyzer, (2) Detailed Scheduler, (3) Factory Model, and (4) Dispatcher that are distributed across many workstations and are connected by a communication network. Concentrating on machine failures, this chapter as well as its later extension is to be the core of the Uncertainty Analyzer module.

6.2. Problem Description

The problem setting in general is a job shop with failure-prone machines. A scheduler decides its production schedule for several weeks ahead. *Jobs* are fulfilled by a make-to-job policy and each job can be of a different part type. A *job* is composed of several number of *units* of one part type. All units in each job of one certain part type goes through several operations according to the designated routing of that particular part type. Besides the number of units and its part type that one job distinguishes itself from others, every job has its own requested arrival date and requested due dates. For the current setting, there is no preemption among jobs.

Machine failures occur from time to time during processing. The mean time between failure and mean duration of failure are known in advance. During the downtime, operators

may be involved with repairing the broken machine, equipping it with proper tools or simply tuning it to a desirable status to let the machine operate again. Downtime is assumed to be in the interrupt-resume regime, that is, once the downtime duration is completed, processing continues at the point of interruption and no rework is required. Consequently, machine failures cause a variation in the processing time and not in the scheduling order sequence.

The problem arises from not knowing when to start an operation and how much time should be considered for an operation given the distribution behavior of the uncertainty is known. Assuming no changeover or setup time, the processing time for a *job* without any machine failure is the *unit* processing time multiplied by the number of units in the job. The units in one job are completed together as the machine resumes whatever operation just before its interrupt. As several interrupts can occur within one particular operation time for a job, its job processing time can be further prolonged. Therefore the completion time is delayed until the very last unit is finished.

The focus of this research is to construct a policy for determining temporal protection that minimizes work-in-process, tardiness and resource idle time, and adheres to the original schedule as much as possible confronting machine uncertainty. We investigate a simpler version of the problem, the single machine scheduling problem, where there are multiple jobs, each only having a single operation, and each requiring the same single machine.

6.3. Related Research

Anthony [Anthony 65] classified the model of control into three broad categories. In brief, these three categories are described as managerial decisions at three hierarchical levels (Hax and Candea [Hax & Candea 84]): (1) **strategic planning**: top level decision of plans for acquisition of resources, (2) **tactical planning**: middle level of plans for utilization of resources, and (3) **operation control**: low level of detailed execution of schedules.

The effects of uncertainty to the manufacturing environment have been investigated at the middle level of the model of control such as well-known analytical approaches to the inventory problem of lot yielding and safety stock [Gerchack et al. 88, Grave 87]. Yet, the temporal deviation from machine failures in the job shop scheduling was not explicitly addressed. Our focus is to examine the effects of uncertainty at the lowest level of operations control and scheduling.

Sources of uncertainty can also be described in these three levels. At the top level, the uncertain market environment can change the product emphasis and labor supply which can in turn change the plan of capacity acquisition. At the middle level, changes in forecast and seasonal demand, yield, raw material quality and quantity can impact the production plan.

And, finally at the lowest level, change of time duration for operations (transition, setup, processing), change of capacity from machine downtime or tool availability and so on can easily invalid a schedule.

The lowest level of operation control provides the day-to-day flexibility needed to meet customer requirements on a daily basis within the guidelines established by the more aggregate plans from the middle level. Taking orders directly from customers, or as generated by the inventory decision system, detailed schedules are drawn up in advance for a week, then a day, and finally to a shift. Decisions at the lowest level are dynamic in nature since at this level a shop faces with various sources of uncertainty at a shorter decision cycle. Unanticipated causes as well as scheduled events contributes to the shop uncertainty at this level of operations control. Raw materials are not always available. Aged tools wear out and affect the precision quality. In particular, machines break down from time to time. Uncertainties in machine performance often cause reality to deviate from schedules. How to tolerate these temporal deviation is the theme of this research.

One of the previous approaches to scheduling at this level is the Sched-star package by Morton et. al. [Morton et al. 86, Morton et al. 88] that dynamically adjusts to the uncertain environment as it can redecide the urgency index for the orders at that time from the imputed (dual) prices of the machines. The dual prices of machines are passed down from an aggregate level for its relative value from these aggregated resources in the inventory and production level. Then, the lowest level can have a narrower focus and perform local optimization. The reactive scheme can be either locally as a recomputing minor price changes as in dispatcher mode, or leaving to rescheduling as in replanning model (Morton [Morton et al. 86]). The prices for this model are computed using heuristics. The disadvantage is that it is hard to return good price estimates under all conditions for heuristics. An analytical approach with a hierarchical control has been studied for a flexible manufacturing environment (Akella et. al. [Akella et al. 84, Akella et al. 87]). The objective of this algorithm is to calculate times at which to dispatch parts into a system in a manner which limits the disruptive effects of machine failures. With this approach, three levels of control are addressed. Its middle level is the major decision level of the scheduler and determines the production rate within capacity limits and achieves the objective function computed off-line by its higher level. Its lowest level decides the actual times at which parts are loaded into the system according to that production rate. However, it is for cumulative demand instead of being addressed to the job shop scheduling. The prior approaches have focused on dynamicly redeciding the urgency index or loading decisions at the time of uncertainty. There are industries with expensive machines so that building predictive schedules with protection allowance ahead is necessary for the day-to-day operation. In our approach, we explicitly take environmental uncertainty into account in order to produce schedules that tolerate temporal deviations and minimize work-in-process and job tardiness.

6.4. Selecting Temporal Protection

Temporal protection specifies when a job is to be released to the factory and the amount of temporal slack to add to the operation's duration. In the following we define how temporal protection is derived and employed in the specification of an operation's reservation for resources. In particular, we focus on how much temporal slack should be added to an operation's duration and how early a job is to be released by determining its earliest start time.

Let the original processing time of an operation be P, which is deterministic in the model, the time between machine failure be a random variable F, and the duration of interrupt be a random variable D. If these two means are known as F and D, then a direct extension of the processing time to include the machine interruptions is given as $P+(P/F)\times D$, where P/F gives the number of interrupts that may occur during the processing and $(P/F)\times D$ gives the total length of the machine downtime. Thus temporal slack is provided in the extended processing time.

Instead of being random variables of known distribution, the duration of the failure and the time between failure may be only known to be bounded approximately. The development of fuzzy number theory has made it possible to express these imprecise informations. Let the bounds are (D_{lb},D_{ub}) for D and (F_{lb},F_{ub}) for F with the means D and F, we can therefore determine the extended processing time using fuzzy algebra [Kaufmann and Gupta (1984)].

- **Fuzzy Number of Type-1** Let A be a number with upper and lower bounds defining a confidence interval noted as $[a_1,a_2]$ where $a_1 \le a_2$. Similarly, let B be a number associated with an interval $[b_1,b_2]$ (where $b_1 \le b_2$) representing an interval of confidence for B.
- **Fuzzy addition**: Assuming two intervals of confidence in real numbers $R\cdot A=[a_1,a_2]$ and $B=[b_1,b_2]$. Hence if $x\in[a_1,a_2]$ and $y\in[b_1,b_2]$, then $x+y\in[a_1+b_1,a_2+b_2]$. Symbolically, we write it as $A(+)B=[a_1,a_2]+[b_1,b_2]=[a_1+b_1,a_2+b_2]$.
- **Fuzzy Subtraction**: $A(-)B=[a_1-b_2,a_2-b_1]$.
- **Fuzzy Multiplication**: $A(\times)B=[a_1\times b_1,a_2\times b_2]$.
- **Fuzzy Division**: $A(/)B=[a_1/b_2,a_2/b_1]$ when A and B are defined in R^+.

From the above fuzzy algebra, the bounds of the extended processing time can be given as follows: $P+P(/)F(\times)D=P+P(\times)D(/)F$ where $D(/)F=[D_{lb}/F_{ub},D_{ub}/F_{lb}]$, so that the *upper* and *lower* bounds of processing time would be: $(P+P\times D_{lb}/F_{ub},P+P\times D_{ub}/F_{lb})$.

The values of Fuzzy bounds may originate from subjective known processing characteristics described by a shop operator, or perhaps from known distributions described by shop statistics. The representation of such a uncertain duration is called a **type-1** fuzzy representation as the bounds are known in advance.

From the knowledge of the domain, the actual processing time is within the specified uncertain bounds. In an extension to representation, we can hypothesize the bounds as **type-2 bounds** similar to the fuzzy number of type-2 [Prade (1979)] representation.

- **Fuzzy number of Type-2:** The lower and upper bounds of an interval of confidence, instead of being ordinary numbers, are fuzzy numbers that themselves have interval of confidence. That is $A=[[a_1,a_1']\,,[a_2,a_2']]$. When $a_1=a_1'$ and $a_2=a_2'$, the interval of confidence of type 2 becomes an interval of **type 1**. If $a_1=a_1'=a_2=a_2'$, we obtain an interval of confidence of **type 0**, an ordinary number.

Oftentimes, there is more uncertainty in a scheduling problem than can be handled by a type-1 bound representation. As a consequence, it is more accurate to represent additional uncertainty; therefore, the upper and lower bounds must reflect uncertainty in their representation as type-2 bounds. The procedure of constructing the **type-2 bounds** is described through the following steps (see Figure 6-1):

1. Use the mean processing time, $P+P\times D/F$, as the inner bound of a type-2 representation. Denote it as p_{inner}.
2. Use the upper bound, $(P+P\times D_{lb}/F_{ub})$, as the outer bound of the type-2 representation. Denote the length of the outer bound as p_{outer}.
3. Divide the slack between p_{outer} and p_{inner} into two segments and denote them as $p_{lower-slack}$ and $p_{upper-slack}$. That is, $p_{lower-slack}=p_{upper-slack}=(p_{outer}-p_{inner})/2=P\times(D_{ub}/F_{lb}-D/F)/2$ as shown in Figure 6-1. The $p_{lower-slack}$ and $p_{upper-slack}$, temporal slacks of equal amount, are designed to protect against uncertainty for possible delays of previous operations or possible delays of consequent operations.

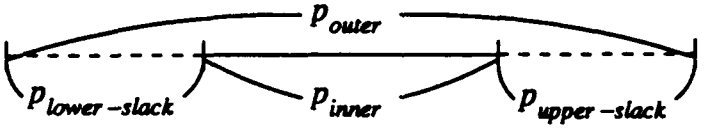


Figure 6-1: Illustration for the type-2 bound

Thus, **type-2 bounds** are constructed with values $A=[[a_1,a_1']\,,[a_2,a_2']]=[[0,p_{lower-slack}],[p_{lower-slack}+p_{inner},p_{outer}]]$ if the operation it represented starts at time 0. Or, $A=[[a_1,a_1']\,,[a_2,a_2']]=[[t,t+p_{lower-slack}],[t+p_{lower-slack}+p_{inner},t+p_{outer}]]$ if it starts at time t. The type-2 bounds of a job can then be used to reserve time block in a schedule for the job.

A numerical example can be used to illustrate the calculations. In the example with intervals and means available through estimations that $(D_{lb},D_{ub})=(12,14)$, $D=13$, $(F_{lb},F_{ub})=(15,25)$, $F=20$ and $P=40$, where all time units are in minutes. The bounds of the protection allowance (including the processing time and interruption estimates) are calculated as: $(P+P\times D_{lb}/F_{ub},P+P\times D_{ub}/F_{lb})=(40+40\times12/25,40+40\times14/15)=(59.2,77.3)$. While using D and F gives us the value of $(P+P\times D/F)=66$. Following the steps, we get

$p_{lower-slack}=p_{upper-slack}=(77.3-66)/2=5.65$. If the operation starts at time 0, the type-2 bounds representation is [[0,5.65],[71.65,77.3]]. The results are shown in Figure 6-2.

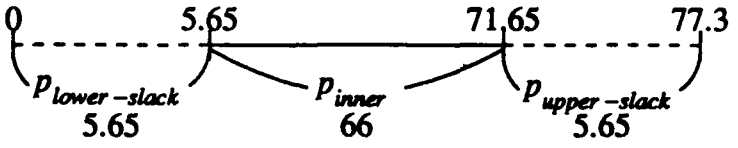


Figure 6-2: Numerical example for the type-2 bound

Thus, extended processing times with temporal allowance for uncertainty can be expressed by type-2 bounds. With these representations at hand we can focus on how to use it as job reservations and how to devise its scheduling method.

6.5. Scheduling with Type-2 Bounds

The difference between using our temporal protection as job reservations in a schedule and using the usual definition of job reservations can be seen as in Figure 6-3 and 6-4. The old reservation is simply a time interval with sharp time points of start time and finish time, while our new definition of reservation is composed of time intervals with the flexibility to start working on a job any time after resource release time and to end a job any time before the latest finish time. The middle segment is the time that guaranteed to have the machine for the job. In the upper and lower segment, the machine is shared between sequential jobs. This flexibility comes from separating the time to release and the time to work on. So, using the new type-2 temporal protection for job reservations, when it is at time t the material for the job is released. If the machine is available, the job will be worked on. If the machine is busy for the previous job, the current job is only sitting idle from this released time. Or, if the machine is still broken down, the job is ready to be picked up whenever the machine gets fixed.

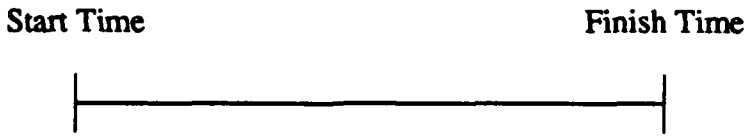


Figure 6-3: Old Job Reservation

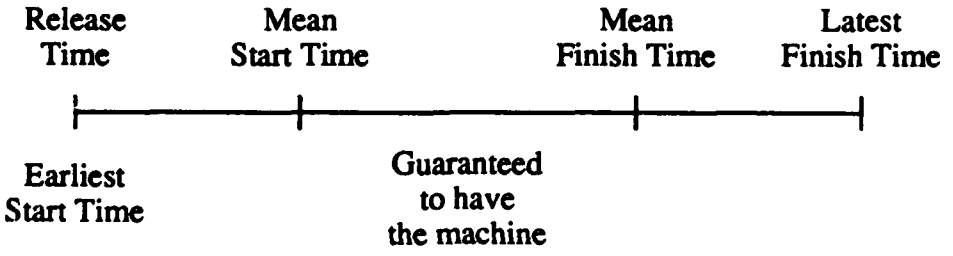


Figure 6-4: New Job Reservation

In scheduling with type-2 bounds, the mean time is used as a middle segment (p_{inner}) to reserve the machine for the operation while the lower and upper segment ($p_{lower-slack}$ or $p_{upper-slack}$) may be overlapped with the protection bounds of other possible consequent jobs. As in Figure 6-5, operation A is designed to be overlapped with operation B. If A completes earlier, then B may start earlier to the extent of its lower bound. If A completes later, then B may start later, again within its bounds. The overlapped section is the $p_{upper-slack}$ part of the operation A and $p_{lower-slack}$ part of the operation B as the bounds are depicted in Figure 6-5. The inner bound, p_{inner} is the reservation to protect the processing for its completion in the bounds. When uncertainty increases, the precision of this predictive schedule is decreased as the slack segments are larger. These overlapped segments give human dispatchers the flexibility to start the job at any time within the slack bounds.

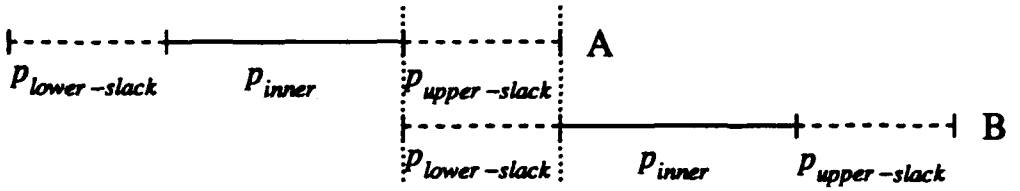


Figure 6-5: Illustration for two overlapped type-2 bound operations

One may imagine that the bounds work as the earliest start time and the latest start time with the earliest finished time and the latest finished times as the activity representation in project management. However, these slacks are designed for a different purpose. They are to protect against uncertainty instead of being arbitrary slacks. Therefore, when time progresses to the point of the lower bound, an operation should start, if the machine is available, to avoid possible subsequent delays in the operation. In project management, slacks exist to indicate the possible earliest start time and the possible finished time and allow a dispatcher to schedule other works in the remaining slack. In the context of uncertainty, the mean processing time (p_{inner}) is reserved for the operation and the slack time ($p_{upper-slack}$ or $p_{lower-slack}$) is reserved for protection against machine uncertainty. For instance, let a job scheduled to start at time t_0 have a type-2 protection allowance as $[[t_0,t_0+p_{lower-slack}],[t_0+p_{lower-slack}+p_{inner},t_0+p_{outer}]]$. At that particular time t_0, the work is immediately released to the shop and the work-in-process time starts counting from time t_0. Therefore, in Figure6-5, the work-in-process time starts from the beginning point of $p_{lower-slack}$ for either operation A or B. Once an operation is ready for processing, a dispatcher should follow the schedule within the prescribed bounds.

In the remaining sections, experiments comparing type-2 bounds with other bounding techniques are described.

6.6. Experiment Design

In order to measure the effectiveness of the type-2 bounds, we compare four methods of determining temporal protection:

1. **type-2 method**: using fuzzy type-2 bounds as $[[t,t+p_{lower-slack}],[t+p_{lower-slack}+p_{inner},t+p_{outer}]]$).
2. **original method**: using the original processing time P as the fixed processing time,
3. **mean method**: using the mean processing time as the fixed processing time (i.e. $P+(P/F)\times D$),
4. **upperbound method**: using the upper bound as the fixed processing time (i.e. using $p_{outer}=P+P\times D_{lb}/F_{ub}$).

These methods were compared in experiments composed of two steps:

1. For each experiment, a schedule was created via dispatch simulation based on a variation of Jackson's algorithm.
2. The schedule was implemented using discrete event simulation in which resource failures occurred.

For each run, data on tardiness, work-in-process and resource idleness was gathered. Experiments were run where the shop load was limited to 50 jobs and 100 jobs. The distribution of the machine failures is simulated by a triangular distribution, so that it has a range of [min, max] and the mode which is the peak of the triangular distribution. Jobs are taken from the OPIS experiments [Chiang et al. 90].

The schedule creation algorithm used in this work is a *dynamic* version of Jackson's algorithm to minimize the maximum lateness [Baker (1974), McMahon and Florian (1975)]. For the static version of the n-job single machine problem, L_{max} is minimized by the sequence of EDD according to Jackson's algorithm. We have it revised to accommodate non-simultaneous job arrivals.

- **Jackson's Algorithm (dynamic version)**: At each job completion the job with the minimum due date b_j among available jobs is selected to begin processing.

  Let S be the set of unscheduled jobs, and a_i be the arrival time, b_i be the due date, and d_i be the duration of job i. Schedule each job as follows:

  1. Set t to 0.
  2. Is there at least one job $i \in S$ such that the arrival time $a_i \leq$ t? If so, go to 4.
  3. Set t=min a_i.
  4. Among all jobs $i \in S$ such that $a_i \leq$ t choose the job j that has the earliest due date b_j; break ties on due date by selecting the job with the largest duration d_j.
  5. Schedule the chosen job next and update t to $t+d_j$,
  6. If S is empty, go to 2. Otherwise, the schedule is complete.

The output of this algorithm is a set of *planned release times* for each job. At a time when the rest of the jobs have all arrived, the dynamic version of the Jackson's algorithm is

equivalent to the static version that is the optimal procedure since all jobs are now simultaneously available.

The adaptation of dynamic Jackson's algorithm is for the methods with the extended processing time assumed to be of fixed length. Only the method of type-2 bounds requires further attention to the overlapped segments, since it is desirable to overlap the uncertain slack segments of consequent operations and undesirable to overlap the reservation segment.

The schedule execution algorithm randomly generates machine failures during the implementation of the schedule, thereby delaying the start of operations, or prolonging the execution of operations. An operation cannot start before its scheduled start time, even if the machine is available. The algorithm is defined as follows:

- Let S be the set of scheduled jobs, and a_i be the arrival time, b_i be the due date, d_i be the duration, and rt_i be the planned release time of job i. Schedule each job as follows:
  1. Set $t = 0$ and machine state to free.
  2. Randomly schedule a machine failure and duration.
  3. If the machine state is free, and the next machine failure time is less than the earliest planned released time of the remaining unexecuted jobs, then set t to the next machine failure time plus its duration, and go to 2.
  4. If the machine state is free, then
     - choose the job i with the earliest planned released time rt_i and remove it from S.
     - If the current time t is less than rt_i, then set $t=rt_i$.
     - Set the completion time $ct=t+d_i$.
     - Set the machine state to busy.
  5. If the machine state is busy, and the next machine failure occurs before the completion time ct, then add the machine failure's duration to the completion time, and go to 2.
  6. Set $t=ct$, set the machine state to free, and go to 2.

6.7. Measuring the Cost of Tardiness and Work-in-Process

Much of the scheduling literature has focused on the optimization of two statistics:

- work-in-process = $\sum$ (actual finish time - planned release time),
- tardiness = $\sum$ (actual finish time - requested due date)$^+$,

where the summations are for all jobs. The first variable is the cost component invested in work-in-process, resulted from the difference between the actual completion time and planned release time. The second dependent variable, is the cost component occurred from not meeting the due date, the absolute value of the subtraction between the actual completion time and requested due date.

In this section we compare the four temporal protection methods along these two dimensions. In order to ascertain the context in which one form of temporal protection out performs another, we compare them on a cost basis, where

Total Cost = C_W * Work-in-Process + C_T * Tardiness

Where C_W and C_T are unit cost of work-in-process and unit cost of tardiness, respectively. Note that we have not included in our cost calculation the cost of idle facilities and personnel in order to contrast the tradeoff between the cost of work-in-process and the cost of tardiness. (In the next section, the cost of idleness will also be included into the total cost.)

Nine sets of cost ratios are used. The first one uses the unit costs of the same magnitude. In cost structures of (2), (4), (6) and (8), the unit cost of work-in-process is less than the unit cost of tardiness. In cost structures of (3), (5), (7) and (9), the unit cost of tardiness is less than the unit cost of work-in-process. In Table 6-1, the total costs are listed while the work-in-process parts are listed inside the parentheses.

Figure 6-6 graphs the costs of the type-2 and upperbound methods alone, since they clearly dominate the original and mean methods. It is clear that savings from using type-2 bounds is significant when unit cost of tardiness is higher than unit cost of work-in-process. That unit cost condition is typical true in manufacturing settings. For other cost structures with a higher unit cost of work-in-process, the upperbound method dominates.

Table of Different Cost Structures for 50 and 100 jobs:

CS	1	2	3	4	5	6	7	8	9
C_W	1	1	1	1	1	2	5	10	20
C_T	20	10	5	2	1	1	1	1	1

Result for 50 jobs:

CS	*Type-2*	*Original*	*Mean*	*Upperbound*
1	620205 (30685)	657155 (57675)	635512 (43712)	746062 (21002)
2	325445 (30685)	357415 (57675)	339612 (43712)	383532 (21002)
3	178065 (30685)	207545 (57675)	191662 (43712)	202267 (21002)
4	89637 (30685)	1176231 (57675)	102892 (43712)	93508 (21002)
5	60161 (30685)	87649 (57675)	73320 (43712)	57255 (21002)
6	90846 (61370)	145324 (115350)	117014 (87424)	78257 (42004)
7	182901 (153425)	318349 (288375)	248150 (218560)	141263 (105010)
8	336326 (306850)	606724 (576750)	466710 (437120)	246273 (210020)
9	643176 (613700)	1183474 (1153500)	903830 (874240)	456293 (420040)

Result for 100 jobs:

CS	Type-2	Original	Mean	Upperbound
1	3449633 (131513)	3854584 (250964)	3492228 (197948)	4790117 (40637)
2	1790573 (131513)	2052774 (250964)	1845088 (197948)	2415377 (40637)
3	961043 (131513)	1151869 (250964)	1021518 (197948)	1228007 (40637)
4	463325 (131513)	611326 (250964)	527376 (197948)	515585 (40637)
5	297419 (131513)	431145 (250964)	362662 (197948)	278111 (40637)
6	428932 (263026)	682109 (501928)	560610 (395896)	318748 (81274)
7	823471 (657565)	1435001 (1254820)	1154454 (989740)	440659 (203185)
8	1481036 (1315130)	2689821 (2509640)	2144194 (1979480)	643844 (406370)
9	2796166 (2630260)	5199461 (5019280)	4123674 (3958960)	1050214 (812740)

Table 6-1: Result for 50 and 100 Jobs

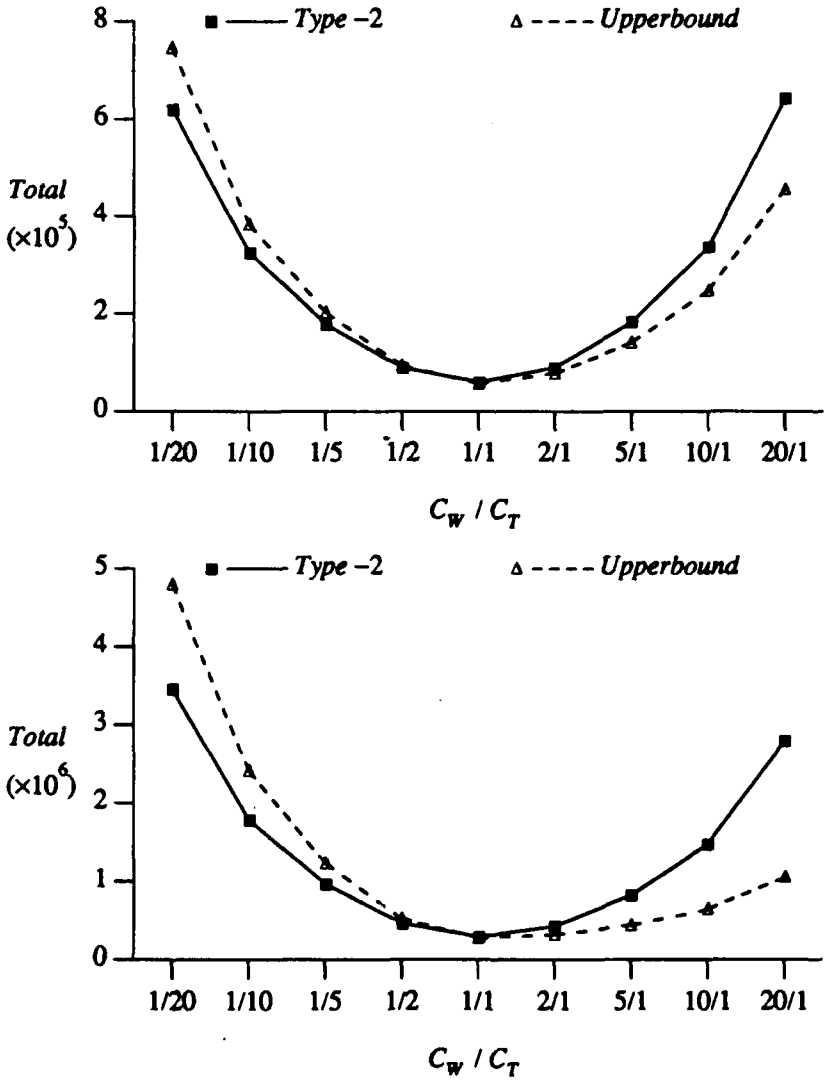


Figure 6-6: Total cost of Type-2 and Upperbound for 50 and 100 jobs

In order to ascertain why type-2 and upperbound protections dominate each other under different cost structures, we will examine a specific scenerio. Table 6-2 summarizes the four protection methods with cost structures work-in-process cost C_W=2 and Tardiness cost C_T=10.

Type-2 protection dominates when Tardiness costs exceed Work-in-Process costs. The reason for this is that the overlapping intervals allow a job to start early when the prior job finishes ahead of schedule, thereby completing the job closer to its requested date. The cost of overlapping is an earlier planned released date makes work-in-process of the type-2 larger than that of the upperbound method. Since Tardiness costs dominate here, the additional work-in-process costs are not large enough to "hurt". The lower total cost for the type-2 method shows that it has the ability of reducing the total cost by the flexibility of the overlapped slack segments.

Upperbound protection dominates when Work-in-Process costs exceed Tardiness costs. Because upperbound durations are large and fixed, the planned release time gets pushed further out into the future for each subsequent job. If a preceding job finishes early, the subsequent job cannot take advantage of the machine availability. Consequently, a resource is usually available at its planned released time. Since upperbound protected jobs are not released early, their Work-in-Process times are low, but since their release times are pushed far into the future, they are usually completed beyond their requested due date. Since Work-in-Process costs dominate here, the additional Tardiness costs are not large enough to "hurt".

D_{lb}	D	D_{ub}	F_{lb}	F	F_{ub}
10	15	20	30	40	50

Case 1: 50-order schedule

Cost	*Type-2*	*Original*	*Mean*	*Upperbound*
work-in-process	30685	57675	43712	21002
tardiness	58952	59948	59180	72506
total cost	89637	117623	102892	93508

Case 2 100-order schedule

Cost	*Type-2*	*Original*	*Mean*	*Upperbound*
work-in-process	131513	250964	197948	40637
tardiness	331812	360362	329428	474948
total cost	463325	611326	527376	515585

Table 6-2: Experiment 1 Result for 50 and 100 Orders

6.8. Measuring Idleness Cost

In the previous section we have seen that the "upperbound" temporal protection method dominated the "type-2" method whenever work-in-process costs are greater than tardiness costs. By increasing the duration of activities to the maximum amount required by the resource failure distributions, later job releases were "pushed out into the future", guaranteeing that they would be worked on when available. But it is often the case that jobs would not require the amount of time specified by the "upperbound" duration. Therefore jobs would finish early, leaving the machine idle until the next job was released. In this section we refine our cost analysis by including a cost for resource idleness.

Resource idleness has many sources. Given an event trace of a schedule execution, there are five points on the time line (figure 6-7) of interest:

1. **Actual Finish Time:** The time the job actually completes in the simulated execution of a schedule.
2. **Planned Finish Time:** The time a job was planned to be completed as determined by a scheduler.
3. **Requested Start Time:** The time a job was requested by the customer to be started time. This is the same as the arrival time.
4. **Planned Release Time:** The time a job is planned by a scheduler to begin work on.
5. **Actual Start Time:** The time the job actually was begun in the simulated execution of a schedule.

In figure 6-7, the dashed line indicates the time between processing two jobs according to the schedule; usually this distance is zero. The solid line indicates the time for processing a job as planned in the schedule. As the actual start time or the actual finished time of the previous job may not be coincide exactly as planned, they are depicted along with the known requested start time of the latter job.

Based on these time points there are three different types of idleness:

1. Total idle due to job arrival = $\sum$(requested start time - actual finished time of previous job)$^+$
2. Total idle due to scheduling = $\sum$(planned release time - max(requested start time, actual finish time))$^+$
3. Total idle due to execution = $\sum$(actual start time - planned release time)$^+$

Only the second type of idleness is due to the temporal protection method. A resource is idled by the scheduling algorithm when it completes an operation and has to wait for the next job to be released to the factory.

The simulations described in the previous section were re-analyzed to include the cost of idleness as follows:

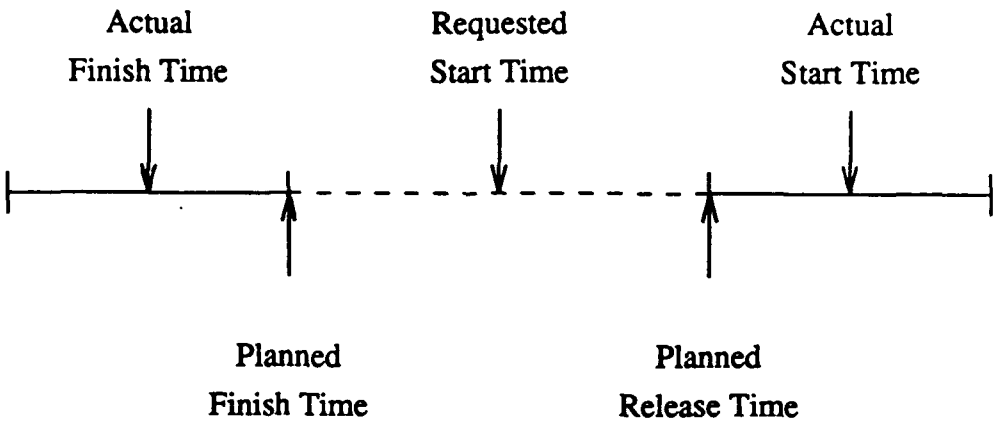


Figure 6-7: Time Components For Idleness

Total Cost=C_W*Work-in-Process + C_T*Tardiness + C_I*Idleness due to scheduling
Where C_W, C_T and C_I are unit cost of work-in-process, unit cost of tardiness and unit cost of idleness respectively.

Figure 6-8 shows the total costs for schedules containing 100 activities and varying the cost of WIP, Tardiness and Idle time between 1, 2 and 5. Going down the X axis, the Idle cost increases from 1 through 5. The figure shows, that as idleness costs increase, type-2 schedules dominate upperbound schedules.

Lets examine a specific run. For the 50 and 100 job cases, we use C_W=1, C_T=2 and C_I=0.5 in Table 6-3.

Result for 50-jobs:

Cost	*Type-2*	*Original*	*Mean*	*Upperbound*
work-in-process	30,685	57,675	43,712	21,002
tardiness	58,952	59,948	59,180	72,506
idleness	22,155	8,662	15,638	30,848
total cost	111,792	126,285	118,530	124,356

Result for 100-jobs:

Cost	*Type-2*	*Original*	*Mean*	*Upperbound*
work-in-process	131,513	250,964	197,948	40,637
tardiness	331,812	360,362	329,428	474,948
idleness	81,160	94,344	125,739	272,845
total cost	544,485	705,670	653,115	788,430

Table 6-3: Total Cost With Idleness for 50 and 100 Jobs

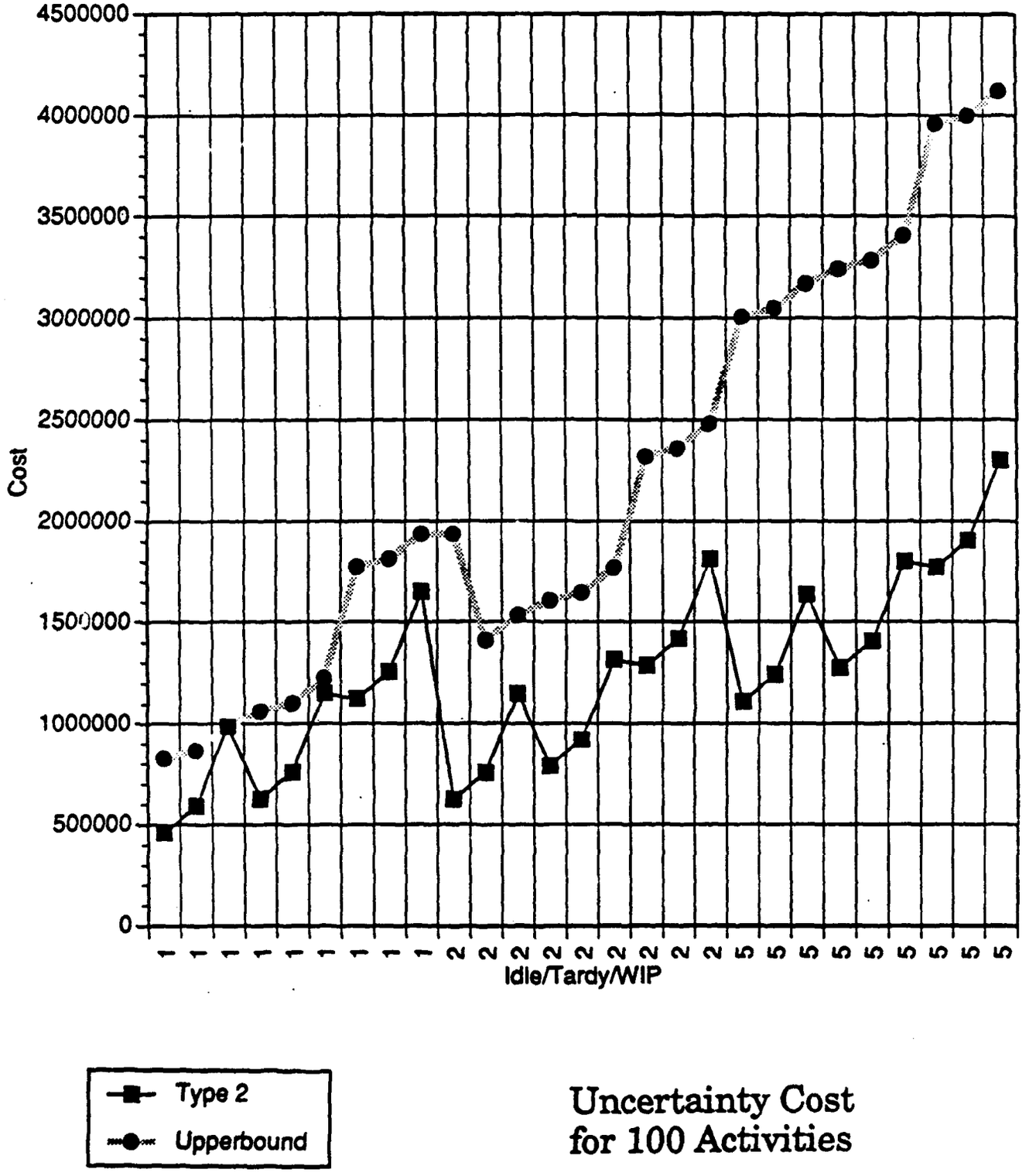


Figure 6-8: Total schedule costs with varying cost coefficients

With the inclusion of an idleness cost at one quarter the cost of tardiness and one half the cost of work-in-process, the "type-2" temporal protection method dominates the other three.

6.9. Conclusion

In domains containing high degrees of uncertainty, whether due to machine failure, resource inavailability or poor performance of tasks, it is necessary to increase the robustness of predictive schedules in order to minimize the costs due to reactive plan repair. In this chapter we have investigated a form of robustness we call ***temporal protection*** which mitigates uncertainty through the manipulation of a job's release date and operation duration. We have introduced the concept of a "type-2" bound as a means of achieving temporal protection. Type-2 temporal protection, by analyzing failure distributions, both increases an operation's duration and releases jobs so that their availability temporally overlaps with other jobs being worked on. In comparison with three other methods that simply increase an operation's duration, we have shown experimentally that the total of three costs: tardiness, work-in-process, and resource idleness, are minimized using the type-2 method. Our current research is extending the results to the multiple machine problem, and to dynamically constructing uncertainty models by monitoring the occurrence of stochastic events.

Chapter 7
Transportation Planning (CDART)

7.1. Overview

The purpose of this section is to describe our work on transportation planning and scheduling as part of the initial phase of the DARPA DART initiative. Our work involved building a prototype software system to demonstrate the utility of applying automatic planning techniques to the problem of revising a TPFDD in the face of changes in scenarios and transportation constraints. The system demonstrated interactive and automatic replanning functions by enabling users to: (1) retrieve and graphically visualize information at different levels of aggregation, (2) augment plans with richer knowledge of constraints and relationships, (3) modify information efficiently through manipulation of aggregate forces, including force ordering and selection of modes and ports, as well as port capacity and lift allocation, (4) analyze the flow of forces and utilization of ports and transportation assets, (5) incrementally assess the feasibility of plan segments, and to accomplish this using an integrated set of tools with short response times. Subsequent sections describe aspects of our work on these capabilities.

7.2. Background

The goal of DART was to demonstrate the potential of advanced technology for dramatically improving the transportation planning process. In particular, DART's goals were to showcase the integration of techniques developed within the artificial intelligence research community with successful analysis and data manipulation tools currently popular in good commercial software. In order to deliver this message effectively, the DART prototype needed to be distinguishable from other current attempts to upgrade planning capabilities. DART also needed to address and go beyond the JOPES user community's consensus for what are obvious incremental improvements to current data processing systems. Therefore the acceptance of DART and the long term planning initiative depended on showcasing the integration of these technologies and the resulting jump in planning capability. The DART prototype achieved these goals by addressing three related levels of planning processes and difficulty that occur in the current data processing environment.

They correspond to (1) TPFDD viewing and manipulation, (2) analysis and hypothesis of TPFDD changes to react to changed constraints, and (3) testing feasibility of TPFDD replanning efforts.

First, it is very difficult to retrieve, visualize and update information from JOPES databases. Even when no analysis is needed to implement a scenario change, planners must spend hours retrieving and updating records manually. Despite the existence of the force module concept, few methods for manipulating units at aggregate levels exist.

Second, the analytic process for generating potential changes to the TPFDD to solve replanning problems is difficult because (1) planners must consider a lot of information and many interacting constraints at one time, and (2) there are few tools which can be used efficiently for viewing and analyzing this information so that decisions can be made which satisfy changed constraints. For example, solving the replanning problem posed by port closing requires the ability to view and analyze port capacities and other constraints, lift constraints, air/sea matchups, temporal constraints on forces, etc. Currently, there are few tools for even manually repeating the cycle of viewing, analyzing, and changing this information to satisfy constraints. There are no tools which help generate potential changes to the TPFDD to solve a problem. There is an equally strong need for supporting replanning by automatic constraint satisfaction (i.e. finding potential replanning alternatives automatically).

Third, planning is difficult because of the slow generate & test cycle corresponding to TPFDD update and feasibility analysis. Feasibility testing is slow and cumbersome, because (1) simulating the effect of each small change requires the same lengthy setup and simulation run of the entire TPFDD and (2) determining which of several alternatives will solve a replanning problem occurs via trial and error. Therefore, there is a need for tools which integrate feasibility testing with other plan analysis and updating tools and which consider alternative replanning solutions automatically. The goal is to reduce or eliminate the gap between generate and test. In some ways, this gap can be eliminated, by merging the analysis process described above and feasibility testing process described here in a single automatic component.

The CDART prototype is a demonstration of capabilities for reducing these difficulties for supported CINC and TRANSCOM staff officers. The prototype illustrated the effect of integrating these automated planning capabilities and their underlying technologies.

7.3. Planning Test Cases

In order to guide our research and development efforts, representatives of US TRANSCOM developed several cases requiring replanning due to changes in conditions or assets of an operation. These cases are outlined below.

Planning Complexity Problem 1

1. Throughput clearance at GABES reduced by 50% due to sabotage by insurgents.
2. Identify all movements going to GABES. Note split shipments.
3. Look for alternate POD that has capacity and that will meet original POD to destination time constraints. Consider SFAX or DJERBA.
4. Determine throughput constraints. If SFAX/DFERBA are maxed out, consider ZARZES or SOUSSE then additional support forces must be inserted into TPFDD. Additionally, ZARZES will require use of self sustaining container ships or a floating crane to be brought in.

Planning Complexity Problem 2

1. Hurricane closes Charleston, Sunny Point and Wilmington.
2. Identify all TPFDD records coming out to those ports. Note split shipments.
3. Look for alternate port; Jacksonville, Florida. Identify throughput capacity and TPFDD records to be changed. Send movements in excess of Jacksonville out of Norfolk. Check split shipments. Modify TPFDD records as necessary.

Planning Complexity Problem 3

1. Extract FM1 from OPLAN 090TP
2. Resequence FM1 to meet show of force LADs.
3. Allocate lift and run model to determine feasibility.
4. Resequence balance of 090TP to reflect fact that FM1 show of force did not deter enemy forces. (NOTE: FM1 is still being moved and there is less lift to move balance of 090TP.) 090TP must now be readied for deployment and transportation analyses. Complete based on revised lift allocations.

7.4. Description of Main Functionalities

CDART will address the different types of planning difficulty described in the previous sections with the following capabilities:

7.4.1. TPFFD Augmentation

A representation was being developed which enabled planners to express knowledge and constraints not contained in standard TPFDDs. These will enable planners to express

- precedence and other relative temporal constraints among forces
- hierarchical relationships among force modules
- aggregation by properties of units (e.g. EAD, LAD, cargo weight, priority)
- deployment priority
- *working* dates, which enable planners to control force flow and evaluate replanning without modifying (and losing sight of) constraints dependent on the original concept of operations as opposed to tentative transportation planning concerns
- port alternatives (e.g. expressing a unit's POE as a set of possible ports)

7.4.2. Database query, update and maintenance

CDART contains a database system built using KnowledgeCraft's (KC) schema representation language (CRL). An ascii TPFDD and related files were converted to a schema database which can be loaded into KC. Three features enabled us to rapidly generate a prototype: (1) the KC context mechanism allows efficient modification and storage of alternative scenarios for what-if analysis, (2) the KC database allows changes to be maintained, and (3) the CRL-Prolog language provides a reasonable approach to query.

We used the context mechanism for rapidly creating one or more working memories for extracting, adding or modifying portions of TPFDDs from a base case. It is also possible to save or load a portion of a TPFDD from/into a working memory context. KC MERGE capabilities enable users to work on different pieces of TPFDD's independently, but still merge them when necessary. We can use this capability to address the extraction and independent deployment of force modules as well as for performing what-if analysis. For practical reasons partly related to cost, computational efficiency, and startup time for implementation, we decided to use CRL-PROLOG rather than ORACLE.

Update capabilities refer to functions for changing fields of TPFDD records efficiently and consistently. Users can refer to sets of records by force module or other search criteria. A command language was developed to change unit fields for the selected set of records. Commands specify absolute values ("set POD to LaGuardia") or relative values (e.g. "set LAD to LAD + 10"; "set LAD to 82nd-Airborne LAD + 3"). Update functions automatically reflect the relationship between force modules and units, (e.g. modifying the LAD's of units will modify the "aggregate" LAD for the force module accordingly).

Finally, transportation files resource data reside in the same working database as

TPFDD's, making it efficient to search and modify force, asset, port and capacity assumptions using the same commands (note that graphical or tabular interfaces provide complementary update capabilities in conjunction with commands). This also provides an effective vehicle for storing scenarios in what-if analysis.

7.4.3. Graphical Displays, Reports and Interfaces

CDART display capabilities were partially provided by some components of CMU's SAGE graphics system, which automatically generates a large variety of displays with limited implementation effort. No attempt was made in CDART to generate maplike displays. Displays and reports include:

- integrated displays of force flow and port capacity
- short-falls analyses by port, unit, mode, and time
- hierarchical displays for viewing and constructing force modules
- comparisons across what-if scenarios (e.g. comparing port throughput or force movement times for the same forces simulated using two different sets of allocation assumptions)
- user controlled generation of charts, graphs, networks and tables of flexible combinations of data retrieved from the TPFDD and other records
- displays supporting constraint analysis (visualizing the most critical units, ports, and assets which are most likely to cause shortfalls).

7.4.4. Automatic planning/replanning capabilities

Displays and reports can be used by planners to interactively assert changes to the TPFDD to achieve replanning goals. However, they can also be used to determine ranges of potential changes which can be considered automatically by CDART's planning/replanning capabilities. CDART capabilities include the automatic reassignment of units to ports and shift of forces temporally in order to minimize shortfalls in the face of changing scenarios. The automatic replanning module considered a minimally disruptive and computationally efficient adjustment to the TPFDD to incorporate a change. Planners have the ability to specify portions of a TPFDD to replan (restricting the system's focus in its search for a solution). Alternatively, CDART will automatically identify a portion of the TPFDD to replan, maintaining the remainder. Replanning here refers to changes in port assignment or force movement times or both.

The replanning tool is designed to enable mechanisms designed in the future to control the reassignment of assets to move requirements. In particular, we anticipate developing replanning strategies for reacting to reductions in port capacity or transportation assets. These strategies partition the TPFDD prior to reconsideration, distinguishing sections to be

preserved and those to reflow or reassign to new ports. The approach is to gradually expand a focus of attention among units in a TPFDD based on the relatedness/degree of coupling of units produced by constraints. The rationale is to reconsider only those units which are most likely to contend for similar resources when replanning is required. For example, when a POD's capacity changes, CDART would attempt to:

- replan only those units potentially displaced by the capacity change,
- replan the latter and units competing for potential alternative ports during the same time-intervals,
- replan the latter and other units competing for lift during the window during which the displaced units can be moved,
- replan the latter and any other units which compete for POE's in the same channel as the POD or alternatives to the POE,
- other intermediate levels of replanning
- .
- .
- TPFDD wide replanning.

Partitioning the TPFDD to focus attention is necessary to alleviate the computational burden (and resulting delay) needed to achieve replanning goals. It has the additional valuable side-effect of reducing TPFDD disruption.

Future research should focus on experimenting with a Constraint Analysis Library containing heuristics for choosing a replanning strategy. Some of these are based on demand density computation similar to those used in the CORTES factory scheduling work and other constraint satisfaction systems. Others are being developed specifically for the transportation problem. The relative effectiveness of these measures can be assessed based on results generated from a preliminary version.

Measures must be developed which identify highly constrained regions of the TPFDD based on combinations of:

- port and lift capacities
- temporal constraints on units
- restrictions on air-sea pairings for split shipments
- port alternatives
- priority (or unit precedence)

Besides focusing the attention of the planning module, these measures can be reported and used by planners to determine additional changes which are most likely to influence closure (these function as suggestions for relaxing otherwise hard constraints). Furthermore, in order to assess the quality of different replanning options, it is necessary to perform some

type of feasibility estimation. As a result, planning must include the task of performing feasibility estimation, and CDART provides users with analysis capabilities. The results are available in forms comparable to port capacity, shortfall and force flow analyses currently available through TFE. The next section defines the transportation problem and bounds on the types and level of detail of constraints handled by the system in the simulator.

7.4.5. Implementation: Hardware & Software Platforms

CDART is implemented in VAX Common Lisp 2.1 using the following features of KnowledgeCraft 3.2: (1) CRL - the frame representation language, (2) CRL-PROLOG for database query purposes, and (3) the window and graphics system for user-system interaction. Database query capabilities were implemented using a combination of CRL-PROLOG and lisp. The choice depended on speed considerations. Working memory management for what-if scenarios will make use of KnowledgeCraft CONTEXT mechanism.

The system was implemented on a microvax 3200 running VMS 4.7 with 48 MEG of main memory.

Finally, our approach to knowledge engineering and system development was to build a preliminary version of the CDART prototype. Our goal was to use this version to obtain feedback on functionalities. It was then advantageous to use the resulting tool to manipulate TPFDD's so as to better define the acceptable manual and automatic solutions to DART sample problems. In fact, we anticipate that this preliminary version will provide a more effective vehicle for knowledge engineering than previously experienced because it will provide fast database retrieval, visualization and modification of TPFDD's and feasibility analysis. As a result, the versions of CDART that were reimplemented as part of the transportation planning initiative are now useful tools for manipulating TPPFDs to construct test cases.

7.5. A Formal Specification of the Air Replanning Problem

The following has been written in an attempt to define more precisely the factors that DART may need to consider in modifying TPFDD records during transportation replanning. The discussion is limited to factors that appear to be important in estimating transportation feasibility. We do not necessarily intend to implement all the equations and tests described, but we do need to understand how variables are defined, and how they interact, at a level of precision greater than that possible in a purely qualitative or verbal account. We expect that most mathematical relationships actually implemented in CDART will be less detailed than the ones given here.

Given the origin, RLD, destination, CRD, and composition of each unit comprising the TPFDD file for an OPLAN, a planner needs to establish the lift mode (air or sea), ports (POE and POD), and intermediate dates (ALD, EAD, LAD) that will produce a transportation feasible plan given a certain lift allocation. To some extent, the movement requirements themselves provide information that can be used to make these decisions. For example,

- mode is related to unit composition
- port selection depends on unit characteristics, origin, destination and mode
- ALD is related to RLD, origin, and POE
- LAD is related to CRD, destination, and POD
- EAD is related to ALD, LAD, and unit composition.

However, there is still latitude in these choices which can be exploited to achieve transportation feasibility.

Fundamentally, the transportation feasibility of a plan depends on the "loads" that will be placed on assets and ports over time by a given set of unit movements. These loads, in turn, depend on the mode, ports, and intermediate dates chosen, and on lower level decisions about how portions of cargo and passengers are allocated to particular assets for movement during particular time periods. Because transportation asset capacity is limited, and so is the capacity of ports to load and unload shipments, all loads must be less than the capacities of the assets and ports they are imposed on, at the time they are imposed. In addition, the set of loads must sum up to the total demand for transportation inherent in the TPFDD.

For any set of loads that meet these conditions, there will be a POD arrival date for each portion of the cargo and passengers comprising each unit. We will call these dates the "calculated arrival dates" (CADs). If the CAD for the last portion of each unit to arrive at its POD is less than or equal to the LAD for that unit, one can say that the OPLAN is "potentially feasible" with respect to the capacities considered and the level of detail used in defining the loads. It is only "potentially" feasible since there may be a finer level of detail at which capacities or other constraints are in fact exceeded for the same set of movement requirements.

We will call the CAD for the last portion of a unit the FAD ("feasible arrival date") for the unit. Technically, a FAD is the hypothetical arrival date for a unit calculated by the feasibility estimation tool called TFE. However, we need some term for the CAD of the last element of a unit, so we hope that no confusion will arise by extending the meaning of FAD to cover this referent regardless of how it is calculated. (Those familiar with TFE will recognize that a "grossly transportation feasible" OPLAN is a potentially feasible OPLAN with respect to the capacities, level of detail, and method of constructing loads used by TFE.)

For replanning, we believe it is essential for DART to have access to

- the demands for transportation inherent in the TPFDD
- the capacities of ports, planes, and/or ships to move units
- the loads placed on these resources in response to the demands
- any other constraints on deployment that need to be respected at the level of analysis being done.

so that appropriate changes to be made in a plan can be identified easily. For instance, if lift allocation decreases, it would be useful to be able to determine exactly which demands and loads are affected and reallocate them to resources or time periods where capacity exists. While this need might be met by using the input and output files to TFE, we think it will be beneficial to have a more dynamic representation of information that can be used to quickly regenerate loads and re-estimate feasibility after changes in a plan have been made.

The following is a first cut at formally defining the information needed in revising the air portion of a transportation plan to meet new demands or constraints. To accommodate the port closure problems specified as DART test cases, we need to include both port constraints and airlift allocation, as is done in TFE. With respect to airlift, we are currently working on a second cut, which will be less detailed than the first cut and will look like the MINOTAUR airlift model extended to include port constraints.

Level of Detail

Any formulation of this problem must adopt some level of detail for describing and computing the interactions between demands, capacities, loads, and other constraints. For the most part, we will use the same level of detail that TFE uses, on the grounds that data at that level of detail is definitely available online somewhere in JOPES. From time to time, we will point out details being left out, approximated, or simplified. For example:

Simplification 1:

> **The basic unit of time for this formulation will be the day. Temporal details at a finer level of granularity, such as the load and unload time for a plane, will generally be ignored.**

Simplification 2:

> **Since this is an air problem, we ignore wet cargo, which always travels by sea, and cargo that is not air-transportable (NAT).**

DEMANDS

The quantities of passengers and cargo associated with a unit are given by:

> **quantity(pax,u), the number of passengers in unit u**
> **quantity(ovr,u), the amount of oversize cargo in u**

quantity(out,u), the amount of outsize cargo in u
quantity(blk,u), the amount of bulk cargo in u

The demand on a port p, of variety v, associated with unit u, on day d is given by:

demand(p,v,u,d) = new demand(p,v,u,d) + carryover demand(p,v,u,d)

where v is one of {pax, ovr, out, blk, TONS}. If v = pax, demand is measured in number of passengers; otherwise it is measured in short tons (stons).

Simplification 3:

We assume that a planner would like all passengers and cargo to arrive at the POD as soon as allowable, namely, on the EAD he has specified.

Hence, for all v and u, the "new demand" on a POD is everything that could permissibly arrive there on EAD(u):

new demand(pod(u),v,u,EAD(u)) = quantity(v,u)

Simplification 4:

We assume that the loading, air shipment, and unloading of cargo or passengers could all be accomplished in a single day using any type of plane and any pair of airports in the absence of capacity constraints.

Hence, for all v and u, the new demand on the POE for a unit is the same as the new demand on its POD:

new demand(poe(u),v,u,EAD(u)) = new demand(pod(u),v,u,EAD(u))

This implies that the ALD and EAD for any unit must be set such that

ALD(u) <= EAD(u)

(If an EAD is ever set to be earlier than an ALD, I can't imagine why.)

Not all of a day's demand is necessarily satisfied on that day. Capacity constraints may limit the amount that can be accommodated. This gives rise to the "carryover demand", which is the amount of demand left over from the previous day:

carryover demand(pod(u),v,u,d) = unsatisfied demand(pod(u),v,u,d-1)
carryover demand(pod(u),v,u,0) = 0

The first day of the plan is day 0, and there is no carry over from a previous day.

The demand that WAS satisfied at a POD on a given day will be called the "load" on that port imposed by u. Of course, the load on a port can only be as great as the demand on any given day:

load(pod(u),v,u,d) <= demand(pod(u),v,u,d)

The unsatisfied demand on a port is then just the difference between the demand that existed and the demand that was satisfied:

```
unsatisfied demand(pod(u),v,u,d) =
                demand(pod(u),v,u,d) - load(pod(u),v,u,d)
```

for all v,u and d.

PORT LOADS

A load on a port is defined to be a portion of port demand that has been chosen for satisfaction on a given day, taking all transportation constraints into account. Across all days, all POD and POE loads associated with a given unit must sum up to the total quantity of passengers and cargo to be moved for that unit:

```
sum[load(poe(u),v,u,d)] = quantity(v,u)
   forall d
sum[load(pod(u),v,u,d)] = quantity(v,u)
   forall d
```

Moreover, it is desired that all loads associated with a unit arrive on or before the LAD for that unit. Hence, if there are several units with demands for the same POD on the same day d, units with LADs <= d must be included in the POD load in preference to units with LADs > d. Mathematically, for any unit u1, and load variety v1 in {pax,ovr,out,blk},

```
if demand(pod(u1),v1,u1,d) > 0 and LAD(u1) > d and
 forsome u2 and v2, unsatisfied-demand(pod(u1),v2,u2,d) > 0
                                                and LAD(u2) <= d,
```

then

```
   load (pod(u1),v1,u1,d) = 0.
```

Now, by simplification 4, any load that goes out of a POE on a given day must also arrive and unload at its POD on the same day. Hence, a load imposed by unit u is the same at both the POD and the POE on the same day:

```
load(pod(u),v,u,d) = load(poe(u),v,u,d)
```

Using this fact, the equations developed above may be summarized as follows:

For p in {poe(u), pod(u)}:

```
demand(p,v,u,d) = new demand(p,v,u,d) + carryover demand(p,v,u,d)
new demand(p,v,u,EAD(u))  = quantity(v,u)
new demand(p,v,u,d)       = 0 for d neq EAD(u)
carryover demand(p,v,u,d) = unsatisfied demand(p,v,u,d-1)
carryover demand(p,v,u,0) = 0
unsatisfied demand(p,v,u,d) = demand(p,v,u,d) - load(p,v,u,d)
```

```
load(p,v,u,d) <= demand(p,v,u,d)
sum[load(p,v,u,d)] =  quantity(v,u)
   forall d
if demand(p,v1,u1,d) > 0 and LAD(u1) > d and
   forsome u2 and v2,
       unsatisfied-demand(p,v2,u2,d) > 0 and LAD(u2) <= d, then
   load (p,v1,u1,d) = 0.
```

TRANSPORTATION RESOURCE CHARACTERISTICS

The characteristics of the airports, aircraft, and lift allocation to be used in a given plan place constraints on the loads that can be imposed on the planes and ports. These have been taken more or less verbatim from TFE.

Characteristics of Each Airport p

ptypes(p)	List of the types of planes that are allowed to use this port.
capacity(p,TONS,d)	Daily cargo thruput capacity, measured in short tons, which can vary by day.
capacity(p,pax,d)	Daily passenger thruput capacity, an integer count, which can vary by day.
capacity(p,MSNS,d)	Maximal number of missions (arrival/departure pairs) that can use this port per day, which can vary by day.

Characteristics of Each Aircraft Type a:

speed(a)	Block speed, in nautical miles/hour.
range(a)	Maximal travel distance without refueling, in nautical miles.
payload(a,out)	Outsize cargo capacity, in stons.
payload(a,ovr)	Oversize cargo capacity, in stons.
payload(a,blk)	Bulk cargo capacity, in stons.
payload(a,pax)	One of: payload(a,pax-c) -- Passenger capacity if some cargo is also aboard, an integer. or payload(a,pax-nc) - Passenger capacity with no cargo aboard, an integer.
ld-time(a)	Time it takes to fully load the plane, in hours.
unld-time(a)	Time it takes to fully unload the plane, in hours.
enrte-delay(a)	Time required for each refueling stop enroute, in hours.

Non-zero values for the different payload types indicate a capability to carry this type of load. The out, ovr, and blk capacities all refer to the same capacity, measured in three different ways. The capacity remaining on a aircraft loaded with L1 outsize cargo stons, L2 oversize stons, and L3 bulk stons for a cargo of variety V in {out,ovr,blk} is given by:

```
(1 - (L1/payload(a,out) + L2/payload(a,ovr) + L3/payload(a,blk))) * V.
```

if a plane has positive payload capacities of each type. If the out,

ovr, or blk payload is zero, a plane cannot carry that type of cargo and the corresponding term is omitted.

(I guess a plane has only a limited number of passenger spaces when it is configured for cargo plus passengers.)

Simplification 5:

> In keeping with simplification 1, we will ignore the range, load time, unload time, and enroute refueling delay times. As will be seen later, this will lead to an over estimate of airlift capacity.

Characteristics of Lift Allocation

Lift allocation is expressed in terms of numbers of planes of various types to be included in a scenario at various times. For each aircraft type a, the allocation specifies:

quantity(a,d)	The number of aircraft of type a to be available on day d.
ute(a,d)	The utilization rate (number of flying hours) to be available on day d for aircraft type a.

PORT THRUPUT CONSTRAINTS ON LOADS

The loads imposed on ports take into account all constraints on those ports. In particular, neither the total cargo load imposed by all units nor the passenger load imposed by all units can exceed the corresponding capacities of any port.

The total cargo load on a port imposed by unit u is:

```
load(p,TONS,u,d) = sum [load(p,v,u,d)]
                          forall v in {ovr,out,blk}
```

The total load of any variety imposed by all units on a particular day is:

```
load(p,v,all-u,d) = sum [load(p,v,u,d]
                          forall u units in the TPFDD
```

So, for p in {poe(u),pod(u)}:

```
load(p,TONS,all-u,d) <= capacity(p,TONS,d)

load(p,pax,all-u,d)  <= capacity(p,pax,d)
```

Before expressing the mission capacity constraint on port loads, we need to define airlift load and capacity.

AIRLIFT MODELING

The cargo and passengers for each unit u must flow through channel chan(u) as they travel from the POE to the POD:

```
chan(u) = c | poe(c) = poe(u) and pod(c) = pod(u)
```

The load on this channel, of variety v, imposed by u is given by:

```
channel-load(chan(u),v,u,d) = load(poe(u),v,u,d)
```

The total load that a channel carries on a given day is the sum of the loads associated with each unit:

```
channel-load(c,v,all-u,d) = sum[load(poe(u),v,u,d)]
                            forall u in the TPFDD with chan(u) = c
```

This is the load that must be carried jointly by the aircraft assigned to c on day d.

In general, channel load may be split up among aircraft of different types, and among individual aircraft within each type, and among individual missions flown by each aircraft. Define

```
msn-load(c,ai,v,m,d)
```

to be the load of variety v carried by plane ai on mission m along channel c on day d. (You were probably wondering where the ai was going to appear in this discussion. Now you know.) Planes assigned to c must be such that:

```
type(ai) is a member of both ptypes(poe(c)) and of ptypes(pod(c)).
```

From the characteristics of aircraft listed earlier, it must also be the case that:

```
payload(type(ai),v) > 0 if msn-load(c,ai,v,m,d) > 0.
```

I.e., only planes that are capable of transporting load variety v are eligible to load with that variety.

Finally, only the number of planes of each type allowed by the lift allocation can be assigned to missions. If identifier ai is of the form <a,i> where a = type(ai) and i = index(ai), then:

```
1 <= index(ai) <= quantity(type(ai),d)
                  when msn-load(c,ai,v,m,d) > 0
```

We know from the characteristics of aircraft that the following constraint:

$$\frac{msn\text{-}load(c,ai,out,m,d)}{payload(type(ai),out)} + \frac{msn\text{-}load(c,ai,ovr,m,d)}{payload(type(ai),ovr)}$$

$$\frac{msn\text{-}load(c,ai,blk,m,d)}{payload(type(ai),blk)} \leq 1.0$$

must hold (where a term is omitted if payload(type(ai),v) = 0), and so must this one:

$$msn\text{-}load(c,ai,pax,m,d) \leq payload(type(ai),pax) =$$
payload(type(ai),pax-c)
if msn-load(c,ai,v,m,d) > 0
 for some v in {ovr,out,blk}
else
 payload(type(ai),pax-nc)

for all missions, channels, planes, and days.

Now in general, a plane may fly more than one mission in a day, where a mission is defined to be a roundtrip flight from a POE to a POD and back to a POE.

Simplification 6:

> Assume that each plane is assigned to a single channel on any given day so that its POEs and PODs are always the same. This is more restrictive than TFE.

The maximum number of missions an aircraft can fly along channel c is then given by:

$$max\text{-}rt(c,ai,d) = floor[ute(type(ai),d) * speed(type(ai))/2 * length(c)]$$

where ute(a,d) * speed(a) is the maximal number of miles the plane can fly in a day and 2 * length(c) is the round trip distance. Since, by simplification 5, we are ignoring any time a plane spends on the ground, we must assume that every plane will fly its maximal number of miles each day if necessary. In other words,

$$\#msns(c,ai,d) \leq max\text{-}rt(c,ai,d)$$

where #msns(c,ai,d) is the number of missions flown by ai on d. This number is further constrained by the mission capacity of the ports on the ends of the channel:

$$sum[\#msns(c,ai,d)] \leq capacity(poe(c),MSNS,d)$$
forall ai

$$sum[\#msns(c,ai,d)] \leq capacity(pod(c),MSNS,d)$$
forall ai

The final constraint on mission loads at this level of detail is:

$$channel\text{-}load(c,v,all\text{-}u,d) = sum[msn\text{-}load(c,ai,v,m,d)]$$
forall ai
for 1 <= m <= #msns(c,ai,d)

for all channels, varieties of load, and days.

Note that any procedure for estimating transportation feasibility that uses this level of detail would have to select a particular combination of ai's and m's for each channel-load. It must further do this using criteria other than feasibility constraints, since there may be many feasible ways to divide channel load into mission loads.

Formal Statement of Transportation Feasibility Criteria

Given a TPFDD and the characteristics of all transportation resources that the units in it use, construct a set of port loads, channel loads, and mission loads that satisfy all of the above defined constraints.

Define FAD(u) = max[d | load(pod(u),v,u,d) > 0], i.e., the latest day on which some portion of unit u's cargo or passengers arrive at their POD.

If, forall u, FAD(u) <= LAD(u), then conclude that the plan is potentially transportation feasible. Otherwise, conclude that the plan is potentially transportation infeasible.

7.6. Representations of Transportation Replanning

The following schema definitions were developed to represent aspects of the transportation replanning problems that required extensions to those represented in TPPFDs.

Notation

The notation used is basically a compressed form of the definitions needed to define CRL (KnowledgeCraft) schemata. The name of each different type of schema is identified as a "type" and the slots for a type are listed following the name. In general, slot descriptors indicate the type(s) of data that can fill the slot (i.e., the range of the slot). If a slot is a relation, the inverse of the relation is shown in *italics*. Meta-slots, if any, and their values, are listed below the corresponding slot, enclosed in angle brackets. A slot which may be empty or contain a single value is listed in square brackets. If there can be zero, one, or several values in a slot, the slot descriptor includes ellipsis dots (...). Otherwise, the slot should contain exactly one value. When the subtypes of a type are intended to be disjoint, the subtypes are listed as "kinds". Otherwise, the subtypes are listed as "is-a+inv".

Schemata

```
type mv-rqmt
        kinds:  force-unit nonunit-rqmt
        pax:          [type is-a: <integer>] <default: 0>
        blk-stons:    [type is-a: <real>]    <default: 0.0>
        blk-mtons:    [type is-a: <real>]    <default: 0.0>
        ovr-stons:    [type is-a: <real>]    <default: 0.0>
        ovr-mtons:    [type is-a: <real>]    <default: 0.0>
```

```
        out-stons:  [type is-a: <real>]    <default: 0.0>
        out-mtons:  [type is-a: <real>]    <default: 0.0>
        nat-stons:  [type is-a: <real>]    <default: 0.0>
        nat-mtons:  [type is-a: <real>]    <default: 0.0>
        pol:        [type is-a: <real>]    <default: 0.0>
        varieties: [type is-a: load-variety ...], variety-of
        n.cargo-cats: [type is-a: <integer>]
        origin: [type is-a: geoloc], origin-of
        rld:    [type is-a: cdate], rld-of
        mode:   [type is-a: travel-mode], mode-of
        mode-char: [type is-a: <atom>]
        origin-chan: [type is-a: channel], origin-channel-for
        poe:    [type is-a: geoloc], poe-of
        ald:    [type is-a: cdate], ald-of
        [alt-poe:[type is-a: geoloc], alt-poe-of]
        pod:    [type is-a: geoloc], pod-of
        ead:    [type is-a: cdate], ead-of
        lad:    [type is-a: cdate], lad-of
        [alt-pod:[type is-a: geoloc], alt-pod-of]
        chan:   [type is-a: channel], channel-for
        priority: [type is-a: <integer>]
        [edd:   [type is-a: cdate], edd-of]
        [fad:   [type is-a: cdate], fad-of]
        destination: [type is-a: geoloc], destination-of
        rdd:    [type is-a: cdate], rdd-of
        destination-chan: [type is-a: channel], destination-channel-for
        cei:    [type is-a: (OR t nil)]
        element-of: [type is-a: force-module ...], elements

type force-unit
        is-a: mv-rqmt
        utc: [type is-a: unit-type], utc-of
        service-code: [type is-a: <atom>]
        org-type: [type is-a: <string>]
        description:  [type is-a: <string>]
        [pic: [type is-a: <atom>]]
        split-code:   [type is-a: <atom>]

type nonunit-rqmt       ;;; there are none of these in our sample TPFDD
        is-a: mv-rqmt
        description:  [type is-a: <string>]
        cargo-code:   [type is-a: <string>]
        supply-class: [type is-a: <string>]
        sq-ft:        [type is-a: <real>]   <default: 0.0>

type unit-type
        utc-of: [type is-a: force-unit ...], utc

type geoloc
        is-a+inv: port
        country/state-code: [type is-a: <atom>]
        origin-of: [type is-a: mv-rqmt...], origin
        destination-of: [type is-a: mv-rqmt ...], destination
        channels-from: [type is-a: channel ...], from
        channels-to:   [type is-a: channel ...], to
        mode-char: [type is-a: <atom>]
        mode:   [type is-a: travel-mode ...], mode-of
```

```
type port
        is-a: geoloc
        element-of: [type is-a: port-group]
        is-a+inv: airport seaport

type port-group
        is-a: set
        elements: [type is-a: port]

type airport
        is-a: port
        mode: air
        max-stons:  [type is-a: real]
        max-pax:    [type is-a: integer]
        max-msns:   [type is-a: integer]
        poe-of:     [type is-a: mv-rqmt ...], poe
        alt-poe-of: [type is-a: mv-rqmt ...], alt-poe
        pod-of:     [type is-a: mv-rqmt ...], pod
        alt-pod-of: [type is-a: mv-rqmt ...], alt-pod


type seaport
        is-a: port
        mode: sea
        max-mtons:  [type is-a: real]
        max-cbbls:   [type is-a: real]
        poe-of:     [type is-a: mv-rqmt ...], poe
        alt-poe-of: [type is-a: mv-rqmt ...], alt-poe
        pod-of:     [type is-a: mv-rqmt ...], pod
        alt-pod-of: [type is-a: mv-rqmt ...], alt-pod
        sea-distances: [type is-a: ], something Don added on

type cdate
        numval: [type is-a: <integer>]
        rld-of: [type is-a: mv-rqmt ...], rld
        ald-of: [type is-a: mv-rqmt ...], ald
        ead-of: [type is-a: mv-rqmt ...], ead
        lad-of: [type is-a: mv-rqmt ...], lad
        edd-of: [type is-a: mv-rqmt ...], edd
        fad-of: [type is-a: mv-rqmt ...], fad
        rdd-of: [type is-a: mv-rqmt ...], rdd

type travel-mode
        kinds: air sea land other-mode
        mode-of: [type is-a: (OR port mv-rqmt channel) ...], mode

type air
        is-a: travel-mode

type sea
        is-a: travel-mode

type land
        is-a: travel-mode

type other-mode
        is-a: travel-mode1
```

```
type channel
        is-a+inv: air-channel sea-channel land-channel other-channel
                null-channel
        origin-channel-for: [type is-a: mv-rqmt ...], origin-chan
        channel-for: [type is-a: mv-rqmt ...], chan
        destination-channel-for: [type is-a: mv-rqmt ...], destination-chan
        pax:     [type is-a: <integer>], sum of pax over all channel units
        mtons:   [type is-a: <real>], sum of all mtons
        stons:   [type is-a: <real>], sum of all stons
        pol:     [type is-a: <real>], sum of all pol
        blk-stons: [type is-a: <real>], sum
        blk-mtons: [type is-a: <real>], sum
        ovr-stons: [type is-a: <real>], sum
        ovr-mtons: [type is-a: <real>], sum
        out-stons: [type is-a: <real>], sum
        out-mtons: [type is-a: <real>], sum
        nat-stons: [type is-a: <real>], sum
        nat-mtons: [type is-a: <real>], sum
        mode-char: [type is-a: <atom>]
        mode:    [type is-a: travel-mode], mode-of
        from:    [type is-a: geoloc], channels-from
        to:      [type is-a: geoloc], channels-to
        length: [type is-a: <real>]


type air-channel
        is-a: channel
        mode: air
        from:  [type is-a: airport]
        to:    [type is-a: airport]

type sea-channel
        is-a: channel
        mode: sea
        from:  [type is-a: seaport]
        to:    [type is-a: seaport]

type land-channel
        is-a: channel
        mode: land
        from:  [type is-a: geoloc]
        to:    [type is-a: geoloc]

type other-channel
        is-a: channel
        mode: other-mode
        from:  [type is-a: geoloc]
        to:    [type is-a: geoloc]

type null-channel
        instance: other-channel
        mode: other-mode
        mode-char: X
        from: [type is-a: geoloc]
        to: [type is-a: geoloc]
        length: 0.0
```

```
type load-variety
        is-a+inv: BLK, OVR, OUT, PAX, POL
        variety-of: [type is-a: mv-rqmt ...], varieties
        ac-preferences: [type is-a: list], best types of planes for cargo
        ship-prefences: [type is-a: list], best types of ships for cargo

type blk
        is-a: load-variety
        ac-preferences: (c-141 c-5 c-130)
        ship-preferences: (ro/ro lash containership breakbulk sea-barge)

type ovr
        is-a: load-variety
        ac-preferences: (c-141 c-5 c-130)
        ship-preferences: (ro/ro lash containership breakbulk sea-barge)

type out
        is-a: load-variety
        ac-preferences: (c-5)
        ship-preferences: (ro/ro lash containership breakbulk sea-barge)

type nat
        is-a: load-variety
        ship-preferences: (ro/ro lash containership breakbulk sea-barge)

type pax
        is-a: load-variety
        ac-preferences: (c-141 c-130 c-5)
        ship-preferences: (ro/ro lash containership breakbulk sea-barge)

type pol
        is-a: load-variety
        ship-preferences: (large-tanker medium-tanker small-tanker)

type force-module
        title-text: [type is-a: <string>]
        description-text: [type is-a: <string>]
        elements: [type is-a: mv-rqmt ...], element-of
        rld: [type is-a: <cdate> <cdate>], max and min of elements
        ald: [type is-a: <cdate> <cdate>], max and min of elements
        ead: [type is-a: <cdate> <cdate>], max and min of elements
        lad: [type is-a: <cdate> <cdate>], max and min of elements
        pax: [type is-a: <integer>], sum of slots in elements
        blk-stons: [type is-a: <real>], sum of slots in elements
        blk-mtons: [type is-a: <real>], sum of slots in elements
        ovr-stons: [type is-a: <real>], sum of slots in elements
        ovr-mtons: [type is-a: <real>], sum of slots in elements
        out-stons: [type is-a: <real>], sum of slots in elements
        out-mtons: [type is-a: <real>], sum of slots in elements
        nat-stons: [type is-a: <real>], sum of slots in elements
        nat-mtons: [type is-a: <real>], sum of slots in elements
        pol: [type: [type is-a: <real>], sum of slots in elements

type aircraft
        speed:          [type is-a: <integer>]
        allocation:     [see below]
        ute:            [see below]
```

```
        out-payload:     [type is-a: <real>]
        ovr-payload:     [type is-a: <real>]
        blk-payload:     [type is-a: <real>]
        paxc-payload:    [type is-a: <integer>]
        paxnc-payload:   [type is-a: <integer>]
        is-a+inv: C-5 C-141 C-130
```

allocation and ute contain a-lists of the form ((d1 . amt1 (d2 . amt2) ...)
where amt1 is the value starting cdate d1, it continues the same for all days between d1 and d2, then switches to the d2 value, amt2, which continues up to d3, etc.)

```
type C-5
        is-a: aircraft
        speed: 438
        ute: ((c000 . 10) (c030 . 8))
        allocation: ((c000 . 70))
        out-payload: 66.5
        ovr-payload: 65.1
        blk-payload: 75.7
        paxnc-payload: 91
        paxc-payload: 8

type C-130
        is-a: aircraft
        speed: 288
        ute: ((c000 . 6) (c005 . 4))
        allocation: ((c000 . 128))
        out-payload: 0.0
        ovr-payload: 12.5
        blk-payload: 13.8
        paxnc-payload: 91
        paxc-payload: 8

type C-141
        is-a: aircraft
        speed: 425
        ute: ((c000 . 10) (c030 . 8))
        allocation: ((c000 . 218))
        out-payload: 0.0
        ovr-payload: 24.5
        blk-payload: 29.6
        paxnc-payload: 153
        paxc-payload: 26

type ship
        speed:       [type is-a: <real>], knots/hr
                     <default: 20.0>
        max-mtons:   [type is-a: <real>], dry cargo capacity in mtons
                     <default: 10000.0>
        mtons/day:   [type is-a: <real>], amount of dry cargo that can
                       be carried onto or off of the ship in one day
                     <default: 5000.0>
        max-pax:     [type is-a: <integer>], passenger capacity
                     <default: 10000>
        pax/day:     [type is-a: <real>], number of passengers that
                       can move onto or off of the ship in one day
```

```
                <default: 5000.0>
        max-cbbls:  [type is-a: <real>], POL capacity in 100s of barrels
                <default: 0.0>
        cbbls/day: [type is-a: <real>], amount of POL that can be
          pumped onto or off of the ship in one day
                <default: 0.0>
        availability:
                [type is-a: <list>]
                [elmt-form:
                (<cdate> <integer> <port-group>)]
                        times and places at which various numbers of ships
          of a given type become available for use, in
          ascending <cdate> order

type breakbulk
        is-a: ship
        speed: 20.5
        max-mtons: 20874.0
        mtons/day: 4156.799
        availability: ((C000 8 EASTCOAST-PORTS) (C002 3 EASTCOAST-PORTS)
                       (C005 1 EASTCOAST-PORTS) (C005 6 GULFCOAST-PORTS)
                       (C010 2 WESTCOAST-PORTS))

type containership
        is-a: ship
        speed: 22.7
        max-mtons: 36988.0
        mtons/day: 36988.0
        availability:((C002 2 EASTCOAST-PORTS) (C005 1 EASTCOAST-PORTS))

type lash
        is-a: ship
        speed: 22.5
        max-mtons: 42042.0
        mtons/day: 56056.0
        availability: ((C002 3 EASTCOAST-PORTS) (C004 1 EASTCOAST-PORTS)
                       (C004 2 GULFCOAST-PORTS))

type ro/ro
        is-a: ship
        speed: 23.5
        max-mtons: 38755.0
        mtons/day: 116256.0
        availability: ((C000 1 EASTCOAST-PORTS) (C002 1 EASTCOAST-PORTS)
                       (C002 1 GULFCOAST-PORTS) (C004 2 EASTCOAST-PORTS)
                       (C004 1 GULFCOAST-PORTS))

type sea-barge
        is-a: ship
        speed: 20.0
        max-mtons: 42400.0
        mtons/day: 101760.0
        availability:((C002 1 EASTCOAST-PORTS) (C004 1 EASTCOAST-PORTS))

type small-tanker
        is-a: ship
        speed: 15.1
```

```
        max-mtons: 0.0
        max-pax: 0
        max-cbbls: 1607.78
        cbbls/day: 1286.223
        availability:((C002 1 EASTCOAST-PORTS) (C004 1 EASTCOAST-PORTS))

type medium-tanker
        is-a: ship
        speed: 16.1
        max-mtons: 0.0
        max-pax: 0
        mtons/day: 0.0
        pax/day 0
        max-cbbls: 3067.8
        cbbls/day 1533.9
        availability: ((C004 2  EASTCOAST-PORTS))

type large-tanker
        is-a: ship
        speed: 20.0
        max-mtons: 0
        max-pax: 0
        mtons/day: 0.0
        pax/day: 0
        max-cbbls: 8928.75
        cbbls/day: 2976.25
        availability: ((C000 1 GULFCOAST-PORTS))
```

7.7. Representing Precedence Among Move Requirements

In order to solve the test case regarding resequencing of TPFDD's in the face of changing conditions (including extracting and constructing deterrent force modules), it was necessary to develop a perspective on precedence constraints among move requirements. The following are the CDART operating hypotheses about how people establish TPFDD dates and a suggestion about how to use them to re-establish dates after changes in force unit sequence or lift allocation.

I. Planners (CINC or component) set "ideal" dates based on inter-unit precedences and travel-time offsets, independent of lift resources.

Suppose there are three related units, u1, u2, u3, that are to be delivered in that order, offset as follows:

```
RDD(u2) =  RDD(u3) - 2
RDD(u1) <= RDD(u2) - 5
```

Unit 2 cannot arrive early at its destination, but unit 1 can. Neither one can arrive late with respect to its successor and still accomplish its part of the u1-u2-u3 mission. (E.g. u1 builds a runway, u2 sets up a control tower, u3 is the landing of planes on the runway).

II. The other dates are set by simple time offsets.

```
(1) LAD(u) = RDD(u) - theater-transit-time(u)
```

(2) EAD(u) = LAD(u) - unit-offload-time+allowable-slack(u)
(3) ALD(u) = RLD(u) + conus-travel-time(u)

These are stated as equalities for the purpose of producing a definite set of transportation requirements, once RDD(u3) is established. The time offsets are, of course, a function of the origin, destination, POE, POD, the size of the unit, ground transportation capabilities, material-handling capabilities at ports, and any activities that the unit must perform during theater-transit (e.g., go down to a seaport and pick up the equipment after a split shipment). I don't know how one might set allowable-slack, but it would be non-zero if the unit could hang around the POD for a while before moving on to the destination without causing a problem.

III. Assume that all RLDs and the unit origins are fixed and constant.

I further hypothesize that planners want to get all units into employment as fast as possible, given the RLDs and subject to the precedence constraints. So they decide on the RDDs for "independent" units (like u3) and let the others fall out according to the equations.

IV. It is required that:

(4) ALD(u) + strategic-travel-time(u) <= EAD(u)

or else the movement requirement is a-priori impossible to satisfy.

The strategic-travel-time is a function of mode and parallelization (i.e, how much of the unit can be shipped out per day).

If (4) is not met, something needs to be done to either:

(a) decrease conus-travel-time(u) (e.g. change POE)
or (b) decrease strategic-travel-time(u) (e.g. go by air)
or (c) change EAD, by either

(i) decreasing unit-offloadtime+allowable-slack
(e.g. offload more in parallel, use a bigger port for a really big unit)
or (ii) decreasing theater-transit-time (e.g. change POD)
or (iii) increasing RDD.

or any combination of the above.

Note that increasing RDD(u3) would require shifting RDD(u2) and RDD(u1) to meet the precedence constraints and the CINC would have to live with the fact that the u1-u2-u3 mission cannot be accomplished before this new RDD(u3). So planners should adjust things until (4) is met (or they will not have done their job).

V. Flow the TPFDD out and iterate on the dates and ports.

So far, my hypothetical planners have not taken lift allocation or competing port usage into account. But a "finished" TPFDD is required to be 'transportation feasible' with respect to a given lift allocation and a given set of port constraints, so I hypothesize that the TPFDD is then flowed out, with TFE, to see what the FADs would be. When units are shown to be late,

planners are forced to change dates and/or ports to obtain feasibility. This leads them to push LADs forward, with RDDs and EADs shifted forward accordingly, and (hopefully) also shifting forward any precedence-dependent units. Then, they rerun to see whether this has created new lateness that has to be eliminated.

Depending on what is late, planners could either shift an entire precedence sequence, or stretch it out. E.g., if only u2 is late in the u1-u2-u3 sequence, one could stretch the time between RDD(u1) and RDD(u2). It may also happen that commanders revise their ideas about what can arrive early, or the amount of time it will take a unit to do its part of the mission (i.e, they change their precedence constraints).

Eventually, if you keep shifting and stretching, you will get a TPFDD that is completely feasible, or feasible enough to satisfy everybody at the TPFDD refinement conference.

VI. Thus, at the end, we have TPFDD dates that combine the effects of precedence and lift constraints.

These hypotheses account for why people think that the remainder of a force can be "closed up" after extracting a deterrent force option. As you observed early on, the RDDs, LADs, and EADs are inflated from what they would be ideally with infinite transportation resources, and can now be deflated.

VII. So how should we automatically close up the remainder of a force?

1. Determine the ideal RDDs for independent units based on minimal offsets of dependent units and given RLD values (or ALD values if you don't worry about conus-travel-time). I think this corresponds roughly to computing the earliest start dates in project management. Units not involved in precedence relations (either because they are truly independent, or because it is not known what the relations are) would be shifted back to their earliest possible RDDs consistent with RLDs and time-offsets.
2. Propagate back to the RDDs for dependent units, and to LADs and EADs for all units, using the precedence relations and time-offset equations.
3. Flow out the TPFDD to determine FADs.
4. Reset all LAD = FAD, propagate backward to reset EADs, and propagate forward to reset RDDs.

Essentially, this automates the non-judgmental parts of the manual process that I hypothesize is used to build TPFDDs in the first place.

As an augmentation to step (1), one could mark certain RDDs as inviolable (i.e., they were real, lift-independent due dates). I would do this by setting a CRD value to be the current RDD value. For these units, one would not reset LAD to FAD after TPFDD flow -- if they are late in this run, they are truly late.

This same procedure could be used to reset TPFDD dates for different lift allocations/port constraints.

Chapter 8
Conclusion

This report summarizes the research performed in Knowledge-Based Logistics Planning. A major contribution during this contract has been the development of a new problem solving paradigm, called Constrained Heuristic Search. Using this paradigm as a basis, we have investigated a number of texture measures with which to construct variable and value ordering heuristics for solving the scheduling problem. The results of this investigation has demonstrated a performance exceeding other approaches, both OR and AI.

As an extension to the texture based scheduling, we have investigated the impact of uncertainty on the precision of predictive and reactive scheduling. We have developed an approach to analyzing uncertainty and using the results to reduce the precision of schedules, so that greater flexibility is given to the reactive (dispatching) component. Experimental results have shown a significant cost reduction when tardiness costs exceed work in process costs.

Much research has been performed towards the creation of a distributed scheduling system. A testbed has been implemented and experiments have been run testing the hypothesis that texture measures can been used as a basis of communication among a distributed set of scheduling agents. We have demonstrated that interactions among agents can be reduced significantly through the communication of constraint graph textures.

Our research in constraint directed planning began half way through the second year of the contract. An interactive, hierarchical, non-linear planning which can extend plan fragments has been constructed as a testbed. This work combined with the HSTS work is being used in the follow on contract to explore the application of texture measures to guide the planning process.

In regards to the CORTES's philosophical position, we can say the following:

Generality Hypothesis: The CHS problem solving paradigm has now been demonstrated in manufacturing scheduling, transportation planning and facility layout. We believe that it is very general.

Flexibility Hypothesis: We have demonstrated our approach for both predictive scheduling and dispatching, and have indications that the same approach can be used in planning.

Uncertainty Hypothesis: In order for plans and schedules to be robust uncertainty must be taken into account. Our work on uncertainty demonstrates that.

Scale Hypothesis: Our work on distributed scheduling indicates that large problems decomposed among agents can be effectively managed using statistical information, i.e., textures.

In conclusion, during the three years, the CORTES project has changed significantly how the Artificial Intelligence approaches solving the scheduling problem. Not only has a new problem solving paradigm been proposed and validated, but the paradigm applies to distributed problem solving as well. In addition, we have shown that we must view the management of uncertainty in a new way in order to construct robust schedules.

Chapter 9

References

)

[Adams et al. 88] Adams, J., Balas, E., and Zawack, D.
The Shifting Bottleneck Procedure for Job Shop Scheduling.
Management Science 34(3):391-401, 1988.

[Akella et al. 84] Akella, R., Choog, Y. and Gershwin, S.
Performance of Hierarchical Production Scheduling Policy.
IEEE transactions on Component Hybrids and manufacturing technology 7(3):225-240, 1984.

[Akella et al. 87] Akella, R. and Singh, M., and Krogh, B.
Hierarchical Control Structures For Failure Prone Multicell Flexible Assembly Systems.
Technical Report, Graduate School of Industrial Administration, Pittsburgh, PA, 1987.

[Alexander 65] Alexander, C.
Notes on the Synthesis of Form.
Harvard University Press, Cambridge MA, 1965.

[Alexander 68] Alexander, C.
A City Is Not a Tree.
Ekistics 139:344-348, 1968.

[Allen 83] Allen, J.F.
Maintaining Knowledge about Temporal Intervals.
Communications of the ACM 26(11):832-843, 1983.

[Anthony 65] Anthony, R.A.
Planning and Control of systems: A frame work for analysis .
Technical Report, Graduate school of Business Administration, Harvard University, Boston, 1965.

[Baker 74] Baker, K.R.
Introduction to Sequencing and Scheduling.
John Wiley & Sons, 1974.

[Baker & Scudder 90]
Baker, K.R., and Scudder, G.D.
Sequencing with Earliness and Tardiness Penalties: A Review.
Operations Research 38(1):22-36, January-February, 1990.

[Baykan & Fox 87]
Baykan, C., and Fox, M.S.
An Investigation of Opportunistic Constraint Satisfaction in Space Planning.
In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 1035-1038. 95 First St., Los Altos, CA 94022, 1987.

[Baykan & Fox 90]
Baykan, C.A., and Fox, M.S.
Constraint Satisfaction Techniques for Spatial Planning.
Intelligent CAD Systems 3: Practical Experience and Evaluation.
In ten Hagen, P., and Veerkamp, P.,
Springer-Verlag, 1990.

[Bitner & Reingold 75]
Bitner, J.R., and Reingold, E.M.
Backtrack Programming Techniques.
Communications of the ACM 18:651-655, 1975.

[Camden et al. 90]
Camden, R., Dunmire, C., Goyal, R., Sathi, N., Elm, B., and Fox, M.S.
Distribution Planning: An Integration of Constraint Satisfaction and Heuristic Search Techniques.
In *Proceedings of the Conference on AI Applications in Military Logistics*. 1990.

[Cammarata et al. 83]
Cammarata, S., McArthur, D., and Steeb, R.
Strategies of cooperation in distributed problem solving.
In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 767-770. IJCAI, Karlsruhe, W. Germany, 1983.

[Chiang & Fox 90]
Chiang, W-Y., and Fox, M.S.
Protection Against Uncertainty In a Deterministic Schedule.
In *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management*. May, 1990.

[Chiang et al. 90] Chiang, W-Y., Fox, M.S., and Ow, P-S.
Factory Model and Test Data Descriptons: OPIS Experiments.
Technical Report CMU-RI-TR-90-05, Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, 1990.

[Collinot et al. 88]Collinot, A., Le Pape, C., and Pinoteau, G.
SONIA: A Knowledge-Based Scheduling Systems.
Artificial Intelligence in Engineering 3(2):86-94, 1988.

[Conry et al. 88] Conry, S., Meyer, R., and Lesser, V.
Multistage Negotiation in Distributed Planning.
In Bond, A. and Gasser, L (editor), *Readings in Distributed AI*. Morgan Kaufmann, 1988.

[Courtois 77] Courtois.
Decomposability.
Academic Press, 1977.

[Davis 87] Davis, E.
Constraint Propagation with Interval Labels.
Artificial Intelligence 32:281-331, 1987.

[Davis & Hamscher 88]
Davis, R., and Hamscher, W.
Model-based Reasoning: Troubleshooting.
Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence.
In Shrobe, H.E.,
Morgan Kaufmann Pub. Inc., 1988, pages 297-346.

[Dechter 89] Dechter, R.
Enhancement Schemes for Constraint Processing: Backjumping, Learning and Cutset Decomposition.
Artificial Intelligence 41:273-312, 1989.

[Dechter & Pearl 87]
Dechter, R. and Pearl, J.
Network-based Heuristics for Constraint-Satisfaction Problems.
Artificial Intelligence 34(1):1-38, 1987.

[deKleer 87] de Kleer, J.
Dependency-Directed Backtracking.
In Shapiro, S. (editor), *Encyclopedia of Artificial Intelligence*. John Wiley and Sons, New York, N. Y., 1987.

[Dincbas et al. 88]
Dincbas, Simonis, and Van Hentenryck.
Solving the Car-Sequencing Problem in CLP.
In *Proceedings of the 1988 European Conference on Artificial Intelligence.* 1988.

[Dunmire et al. 90]
Dunmire, C., Sathi, N., Goyal, R., Fox, M., and Kott, A.
Ammunition Inventory Planning: An Integration of Configuration and Resource Allocation Techniques.
In *Proceedings of the Conference on AI Applications in Military Logistics.* 1990.

[Durfee 87] Durfee, E.H.
A Unified Approach to Dynamic Coordination: Planning Actions and Interactions in a Distributed Problem Solving Network.
PhD thesis, COINS, University of Massachusetts, 1987.

[Durfee et al. 85] Durfee, E., Lesser, V., and Corkill, D.
Coherent Cooperation Among Communicating Problem Solvers.
Technical Report, Department of Computer Science and Information Science, University of Massachusetts - Amherst, Massachusetts 01003, September, 1985.

[Eastman 69] Eastman, C.
Cognitive Processes and Ill-defined Problems.
In *Proceedings of the International Joint Conference on Artificial Intelligence.* 1969.

[Elleby et al. 88] Elleby, P., Fargher, H.E., and Addis, T.R.
Reactive Constraint-Based Job-Shop Scheduling.
Expert Systems and Intelligent Manufacturing.
In Oliff, M.D.,
Elesevier Science Publishing Co., 1988, pages 1-10.

[Erman et al. 80] Erman, L.D., Hayes-Roth, F., Lesser, V.R., and Reddy, D.R.
The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty.
ACM Computing Surveys 12(2):213-253, 1980.

[Etherington et al. 89]
Etherington, D. W., Borgida, A., Brachman, R. J., and Kautz, H.
Vivid Knowledge and Tractable Reasoning: Preliminary Report.
In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1146-1152. 1989.

[Fikes & Nilsson 71]
Fikes, R.E., and Nilsson, N.S.
Strips: A New Approach to the Application of Theorem Proving to Problem Solving.
Artificial Intelligence 2:189-208, 1971.

[Fisher 76] Fisher, M.L.
A Dual Algorithm for the One Machine Sequencing Problem.
Mathematical Programming 11:229-251, 1976.

[Fox 81] Fox, M.S.
An Organizational View of Distributed Systems.
IEEE Transactions on Systems, Man, and Cybernetics SMC-11(1):70-80, 1981.

[Fox 83] Fox, M.S.
Constraint-Directed Search: A Case Study of Job-Shop Scheduling.
PhD thesis, Carnegie Mellon University, 1983.
CMU-RI-TR-85-7, Intelligent Systems Laboratory, The Robotics Institute, Pittsburgh,PA.

[Fox 87] Fox, M.S.
Constraint-Directed Search: A Case Study of Job-Shop Scheduling.
Morgan Kaufmann Publishers, Inc., 1987.

[Fox 90] Fox, M.S.
Constraint Guided Scheduling: A Short History of Scheduling Research at CMU.
Computers and Industry 14:79-88, 1990.

[Fox & Sadeh 90] Fox, M.S., and Sadeh, N.
Why Scheduling is Difficult: A CSP Perspective.
In *Proceedings of the European Conference on Artificial Intelligence*, pages 754-767. 1990.

[Fox & Smith 84] Fox, M.S., and Smith, S.F.
ISIS: A Knowledge-Based System for Factory Scheduling.
Expert Systems 1(1):25-49, 1984.

[Fox & Sycara 90]Fox, M.S. and Sycara, K.
Overview of CORTES: A Constraint Based Approach to Production Planning, Scheduling and Control.
In *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management*. May, 1990.

[Fox et al. 89] Fox, M.S., Sadeh, N., and Baykan, C.
Constrained Heuristic Search.
In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 309-316. Morgan Kaufmann Pub. Inc., 1989.

[Frederking & Chase 90]
Frederking, R.E., and Chase, L.L.
Planning in a CIM Environment: Research Towards a Constraint-Directed Planner.
In *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management*. May, 1990.

[French 82] French, S.
Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop.
J. Wiley & Sons, 1982.

[Freuder 82] Freuder, E.C.
A Sufficient Condition for Backtrack-free Search.
Journal of the ACM 29(1):24-32, 1982.

[Fry et al. 87] Fry, T., Leong, G.K., and Rakes, T.R.
Single Machine Scheduling: A Comparison of Two Solution Procedures.
OMEGA International Journal of Management Science 15(4):277-282, 1987.

[Garey & Johnson 79]
Garey, M.R., and Johnson, D.S.
Computers and Intractability: A Guide to the Theory of NP-Completeness.
Freeman and Co., 1979.

[Garey et al. 88] Garey, M.R., Tarjan, R.E., and Wilfgong, G.T.
One-Processor Scheduling with Symmetric Earliness and Tardiness Penalties.
Mathematics of Operations Research 13(2):330-348, May, 1988.

[Gaschnig 79] Gaschnig, J.
Performance Measurement and Analysis of Certain Search Algorithms.
Technical Report CMU-CS-79-124, Computer Science Department, Carnegie Mellon University, 1979.

[Gerchack et al. 88]
Gerchak, Y., Vickson, R.G., and Parlar, M.
Periodic Review Production Models with Variable Yield and Uncertain Demand.
IIE Transactions 20(2):91-103, 1988.

[Ginsberg 89] Ginsberg, M.L.
Universal Planning: An (Almost) Universally Bad Idea.
AI Magazine 10(4):40-44, 1989.

[Goldratt 80] Goldratt, E.M.
Optimized Production Timetable: Beyond MRP: Something Better is finally Here.
October, 1980
Speech to APICS National Conference.

[Golomb & Baumert 65]
Golomb, S.W., and Baumert, L.D.
Backtrack Programming.
Journal of the Association for Computing Machinery 12(4):516-524, 1965.

[Grave 87] Graves, S.
Safety Stock in Manufacturing Systems.
Technical Report, Sloan School of Management, MIT, Cambridge, 1987.

[Haralick & Elliott 80]
Haralick, R.M., and Elliott, G.L.
Increasing Tree Search Efficiency for Constraint Satisfaction Problems.
Artificial Intelligence 14(3):263-313, 1980.

[Hax & Candea 84]
Hax, A., and Candea, D.
Production and Inventory Management.
Prentice-Hall, New Jersey, 1984.

[Hayes-Roth et al. 79]
Hayes-Roth, B., Hayes-Roth, F., Rosenchein, S., and Cammarata, S.
Modeling Planning as an Incremental, Opportunistic Process.
In *Proc. IJCAI-79*, pages 375-383. 1979.

[Ishida et al. 90] Ishida, T., Yokoo, M, and Gasser, L.
An Organizational Approach to Adaptive Production Systems.
In *Proceedings of AAAI-90, Boston, Mass.*. 1990.

[Jacobs 84] Jacobs, F.R.
OPT Uncovered: Many Production Planning And Scheduling Concepts Can Be Applied With Or Without The Software.
Industrial Engineering 16(10):32-41, October, 1984.

[Kaufmann & Gupta 85]
Kaufmann, A. and Gupta, M.
Introduction to Fuzzy Arithmetic: Theory and Applications.
Van Nostrand Reinhold, New York, N.Y., 1985.

[Krishnan et al. 90]
Krishnan, V., Navinchandra, D., Rane, P., and Rinderle, J.R.
Constraint Reasoning and Planning in Concurrent Design.
Technical Report, Robotics Institute, Carnegie Mellon University, 1990.
CMU-RI-TR 90-03.

[Kuwabara & Lesser 89]
Kuwabara, K., and Lesser, V.
Extended Protocol for Multi-Stage Negotiation.
In *Proceedings of the 9th International Workshop on DAI.* 1989.

[Laird et al. 87] Laird, J.E., Newell, A., and Rosenbloom, P.S.,
SOAR: An Architecture for General Intelligence.
Artificial Intelligence 33(1):1-64, September, 1987.

[LePape & Smith 87]
LePape, C. and Smith, S.F.
Management of Temporal Constraints for Factory Scheduling.
In C. Rolland, M. Leonard, and F. Bodart (editors), *Proceedings IFIP TC 8/WG 8.1 Working Conference on Temporal Aspects in Information Systems (TAIS 87).* Elsevier Science Publishers, held in Sophia Antipolis, France, May, 1987.

[Lesser 90] Lesser, V.
An Overview of DAI: Viewing Distributed AI as Distributed Search.
Journal of the Japanese Society of Artificial Intelligence 5(4), 1990.

[Levesque 86] Levesque, H.
Making Believers Out of Computers.
Artificial Intelligence 30:81-108, 1986.

[Mackworth 77] Mackworth, A.K.
Consistency in Networks of Relations.
Artificial Intelligence 8(1):99-118, 1977.

[Mackworth 87] Mackworth, A.K.
Constraint Satisfaction.
Encyclopedia of Artificial Intelligence.
In Shapiro, S.,
J. Wiley & Son, 1987.

[Mackworth & Freuder 85]
Mackworth, A.K., and Freuder, E.C.,
The Complexity of some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems.
Artificial Intelligence 25(1):65-74, 1985.

[Minton et al. 90] Minton, S., Johnston, M.D., Philips, A.B., and Laird, P.
Solving Large-Scale Constraint Satisfaction and Scheduling Problems Using a Heuristic Repair Method.
In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 17-24. MIT Press, 1990.

[Montanari 74] Montanari, U.
Networks of Constraints.
Proc. IFIP Congress :727-732, 1974.

[Morton et al. 86] Morton, T.E., Lawrence, S.R., Rajagopalan, S., and Kekre, S.
MRP-Star: PATRIARCHs Planning Module.
Technical Report, Carnegie-Mellon University, 1986.

[Morton et al. 88] Morton, T.E., Lawrence, S.R., Rajagopalan, S., and Kekre, S.
Sched-Star: A Price-Based Shop Scheduling Module.
Journal of Manufacturing and Operations Management :131-181, 1988.

[Muscettola & Smith 87]
Muscettola, N., and Smith, S.F.
A Probabilistic Framework for Resource-Constrained Muti-Agent Planning.
In *Proceedings of the Tenth International Conference on Artificial Intelligence*, pages 1063-1066. 1987.

[Nadel 86] Nadel, B.A.
The General Consistent Labeling (or Constraint Satisfaction) Problem.
Technical Report DCS-TR-170, Department of Computer Science, Laboratory for Computer Research, Rutgers University, New Brunswick, NJ 08903, 1986.

[Nadel 88] Nadel, B.A.
Tree Search and Arc Consistency in Constraint Satisfaction Algorithms.
Search in Articial Intelligence.
In Kanal, L., and Kumar, V.,
Springer-Verlag, 1988.

[Newell 69] Newell, A.
Heuristic Programming: Ill-Structured Problems.
Progress in Operations Research :360-414, 1969.

[Newell 73] Newell, A.
Artificial Intelligence and the Concept of Mind.
Computer Models of Thought and Language.
In Schank, R.C., and Colby, K.M.,
W.H. Freeman and Co., 1973.

[Newell & Simon 76]
Newell, A., and Simon, H.A.
Computer Sciences as Empirical Inquiry: Symbols and Search.
Communications of the ACM 19(3):113-126, 1976.

[Nilsson 89] Nilsson, N.J.
Action Networks.
In *Proceedings from the rochester Planning Workshop: From formal systems to Practical Systems.* University of Rochester, Rochester, NY, 1989.

[Ow 85] Ow, P-S.
Focused Scheduling in Proportionate Flowshops.
Management Science 31(7):852-869, 1985.

[Ow & Morton 89]
Ow, P-S., and Morton, T.
The Single Machine Early/Tardy Problem.
Management Science 35(2):177-191, 1989.

[Ow & Smith 88] Ow, P-S., and Smith, S.F.
Viewing Scheduling as an Opportunistic Problem-Solving Process.
Annals of Operations Research 12:85-108, 1988.

[Ow et. al. 88] Ow, P-S., Smith, S.F., and Thiriez, A.
Reactive Plan Revision.
In *Proceedings AAAI-88.* St. Paul, Minnesota, August, 1988.

[Panwalkar & Iskander 77]
Panwalkar, S.S., and Iskander, W.
A Survey of Scheduling Rules.
Operations Research 25(1):45-61, 1977.

[Parunak et al. 86]
Parunak, H.V., Lozo, P.W., Judd, R., and Irish, B.W.
A Distributed Heuristic Strategy for Material Transportation.
In *Proceedings 1986 Conference on Intelligent Systems and Machines.* Rochester, Michigan, 1986.

[Prade 79] Prade, H.
Using fuzzy set theory in a scheduling problem.
Fuzzy Sets and Systems 2(2):153-165, 1979.

[Prosser 89] Prosser, P.
A Reactive Scheduling Agent.
In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligent,* pages 1004-1009. Morgan Kaufmann Pub. Inc., 1989.

[Reitman 65] Reitman, W.
Cognition and Thought.
Wiley, 1965.

[Rinnooy Kan 76] Rinnooy Kan, A.H.G.
Machine Scheduling Problems: Classification, complexity, and computations.
PhD thesis, University of Amsterdam, 1976.

[Roth & Mattis 88]
Roth, S.F., and Mattis, J.
Data Characterization for Intelligent Graphics Presentation.
In *Proceedings of the CHI '90 Conference.* ACM, Seattle, Washington, 1988.

[Sacerdoti 74] Sacerdoti, E.D.
Planning in a Hierarchy of Abstraction Spaces.
Artificial Intelligence 5(2):115-135, 1974.

[Sacerdoti 77] Sacerdoti, E.
A Structure for Plans and Behavior.
Elsevier, North-Holland, New York, 1977.

[Sadeh 91] Sadeh, N.
Look-ahead Techniques for Micro-opportunistic Job Shop Scheduling.
PhD thesis, School of Computer Science, 1991.

[Sadeh & Fox 88] Sadeh, N., and Fox, M.S.
Preference Propagation in Temporal Constraints Graphs.
Technical Report, Intelligent Systems Laboratory ,The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, 1988.
CMU-RI-TR-89-2.

[Sadeh & Fox 89] Sadeh, N. and Fox, M.S.
Focus of Attention in an Activity-based Scheduler.
In *Proceedings of the NASA Conference on Space Telerobotics.* 1989.

[Sadeh & Fox 90a]
Sadeh, N., and Fox, M.S.
Variable and Value Ordering Heuristics for Activity-Based Job-Shop Scheduling.
In *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management.* May, 1990.

[Sadeh & Fox 90b]
Sadeh, N., and Fox, M.S.
Generating Advice for Hard Constraint Satisfaction Problems: An Application to Job-Shop Scheduling.
Technical Report, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, 1990.
In Preparation.

[Sathi et al. 85] Sathi, A., Fox, M.S., and Greenberg, M.
Representation of Activity Knowledge for Project Management.
IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-7(5):531-552, September, 1985.

[Schoppers 89] Schoppers, M.J.
In Defense of Reaction Plans as Caches.
AI Magazine 10(4):51-60, 1989.

[Serafini et al. 88]
Serafini, P., Ukovich, W.,Kirchner, H.,Giardina, F., and Tiozzo, F.
Job-shop scheduling: a case study.
Operations Research Models in FMS.
In F. Archetti, M. Lucertini, and P. Serafini,
Springer, Vienna, 1988.

[Silver & Peterson 85]
Silver, E.E., and Peterson, R.
Decision Systems for Inventory Management and Production Planning.
Wiley, 1985.

[Simon 73] Simon, H.A.
The Structure of Ill-Structured Problems.
Artificial Intelligence 4:181-200, 1973.

[Smith & Hynynen 87]
Smith, S.F. and Hynynen, J.E.
Integrated Decentralization of Production Management: An Approach for Factory Scheduling.
In *Proceedings ASME Annual Winter Conference: Symposium on Integrated and Intelligent Manufacturing.* Boston, MA, December, 1987.

[Smith & Ow 85] Smith, S.F., and Ow, P-S.
The Use of Multiple Problem Decompositions in Time Constrained Planning Tasks.
In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1013-1015. 95 First St., Los Altos, CA 94022, 1985.

[Smith et al. 86] Smith, S., Fox, M.S., and Ow, P-S.
Constructing and Maintaining Detailed Production Plans: Investigations into the Development of Knowledge-Based Factory Scheduling Systems.
AI Magazine 7(4):45-61, Fall, 1986.

[Srinivasan 71] Srinivasan, V.
A Hybrid Algorithm for the One Machine Sequencing Problem to Minimize Total Tardiness.
Naval Res. Logist. Quart. 18:317-327, 1971.

[Stefik 81] Stefik, M.
Planning with Constraints (MOLGEN: Part 1).
Artificial Intelligence 16(2):111-140, 1981.

[Sycara et al. 90] Sycara, K., Roth, S., Sadeh, N., and Fox, M.S.
Distributing Production Control.
In *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management.* May, 1990.

[Sycara et al. 91] Sycara, K., Roth, S., Sadeh, N., and Fox, M.S.
Resource Allocation in Distributed Factory Scheduling.
IEEE Expert 6(1):29-40, 1991.

[van de brug et al. 86]
van de Brug, A., Bachant, J., and McDermott, J.
The Taming of R1.
IEEE Expert 1(3):33-42, 1986.

[Veloso 89] Veloso, M.M.
NoLimit - The nonlinear problem solver for Prodigy: User's and programmer's manual.
Technical Report CMU-CS-89-210, School of Computer Science, Carnegie Mellon University, 1989.

[Vepsalainen & Morton 87]
Vepsalainen, A.P.J., and Morton, T.E.
Priority Rules for Job Shops with Weighted Tardiness Costs.
Management Science 33(8):1035-1047, 1987.

[Waterman & Hayes-Roth 78]
Waterman, D.A., and Hayes-Roth, F.
An Overview of Pattern-Directed Inference Systems.
Pattern-Directed Inference Systems.
In Hayes-Roth, F., Waterman, D.A., and Lenat, D.B.,
Addison Wesley Pub. Co., 1978.

[Wilkins 88] Wilkins, D.E.
Practical Planning.
Morgan Kaufmann Publishers Inc., 1988.

[Yokoo et al. 90] Yokoo, M., Ishida, T., and Kuwabara, K.
Distributed Constraint Satisfaction for DAI Problems.
In *Proceedings of the 10th International Workshop on DAI.* Banderra, Texas, 1990.

[Zadeh 65] Zadeh, L.A.
Fuzzy Sets.
Information Control 23:338-353, 1965.

DISTRIBUTION LIST

addresses	number of copies
RL/C3C ATTN: Northrup Fowler III Griffiss AFB NY 13441-5700	10
Carnegie Mellon University ATTN: Mark S. Fox 5000 Forbes Ave Pittsburgh PA 15213	5
RL/DOVL Technical Library Griffiss AFB NY 13441-5700	1
Administrator Defense Technical Info Center DTIC-FDAC Cameron Station Building 5 Alexandria VA 22304-6145	2
Defense Advanced Research Projects Agency 1400 Wilson Blvd Arlington VA 22209-2308	2
RL/C3AB Griffiss AFB NY 13441-5700	1
HQ USAF/SCTT Washington DC 20330-5190	1
SAF/AQSC Pentagon Rm 4D 269 Wash DC 20330	1

Naval Warfare Assessment Center 1
GIDEP Operations Center/Code 30G
ATTN: E Richards
Corona CA 91720

HQ AFSC/XTH 1
Andrews AFB MD 20334-5000

HQ SAC/SCPT 2
OFFUTT AFB NE 68046

HQ TAC/DRIY 1
ATTN: Maj. Divine
Langley AFB VA 23665-5575

HQ TAC/DOA 1
Langley AFB VA 23665-5554

ASD/ENEMS 1
Wright-Patterson AFB OH 45433-6503

SM-ALC/MACEA 1
ATTN: Danny McClure
Bldg 237, MASOF
McClellan AFB CA 95652

WRDC/AAAI-4 1
Wright-Patterson AFB OH 45433-6543

WRDC/AAAI-2 1
ATTN: Mr Franklin Hutson
WPAFB OH 45433-6543

AFIT/LDEE 1
Building 642, Area B
Wright-Patterson AFB OH 45433-6583

WRDC/MTEL 1
Wright-Patterson AFB OH 45433

AAMRL/HE 1
Wright-Patterson AFB OH 45433-6573

Air Force Human Resources Lab 1
Technical Documents Center
AFHRL/LRS-TDC
Wright-Patterson AFB OH 45433

AUL/LSE 1
Bldg 1405
Maxwell AFB AL 36112-5564

HQ AFSPACECOM/XRA 1
STINFO Officer
ATTN: Dr. W. R. Matoush
Peterson AFB CO 80914-5001

HQ ATC/TTOI 1
ATTN: Lt Col Killian
Randolph AFB TX 78150-5001

AFLMC/LGY 1
ATTN: Maj. Shaffer
Building 205
Gunter AFS AL 36114-6693

US Army Strategic Def 1
CSSD-IM-PA
PO Box 1500
Huntsville AL 35807-3801

Ofc of the Chief of Naval Operation
ATTN: William J.Cook
Navy Electromagnetic Spectrum Mgt
Room 5A678, Pentagon (OP-941)
Wash DC 20350 — 1

Commanding Officer
Naval Avionics Center
Library D/765
Indianapolis IN 46219-2189 — 1

Commanding Officer
Naval Ocean Systems Center
Technical Library
Code 96428
San Diego CA 92152-5000 — 1

Cmdr
Naval Weapons Center
Technical Library/C3431
China Lake CA 93555-6001 — 1

Superintendent
Code 524
Naval Postgraduate School
Monterey CA 93943-5000 — 1

Space & Naval Warfare Systems Comm
Washington DC 20363-5100 — 1

CDR, U.S. Army Missile Command
Redstone Scientific Info Center
AMSMI-RD-CS-R/ILL Documents
Redstone Arsenal AL 35898-5241 — 2

Advisory Group on Electron Devices
201 Varick Street, Rm 1140
New York NY 10014 — 2

Los Alamos National Laboratory
Report Library
MS 5000
Los Alamos NM 87544 — 1

AEDC Library
Tech Files/MS-100
Arnold AFB TN 37389

1

Commander, USAG
ASQH-PCA-CPL/Tech Lib
Bldg 61801
Ft Huachuca AZ 85613-6000

1

1839 EIG/EIT
Keesler AFB MS 39534-6348

1

AFEWC/ESRI
San Antonio TX 78243-5000

3

ESD/XRR
Hanscom AFB MA 01731-5000

1

SEI JPO
ATTN: Major Charles J. Ryan
Carnegie Mellon University
Pittsburgh PA 15213-3890

1

Director NSA/CSS
T5122/TDL
ATTN: D W Marjarum
Fort Meade MD 20755-6000

1

Director NSA/CSS
W157
9800 Savage Road
Fort Meade MD 21055-6000

1

NSA
ATTN: D. Alley
Div X911
9500 Savage Road
Ft Meade MD 20755-6000 1

Director
NSA/CSS
W11 DEFSMAC
ATTN: Mr. Mark E. Clesh
Fort George G. Meade MD 20755-6000 1

Director
NSA/CSS R12
ATTN: Mr. Dennis Heinbuch
9800 Savage Road
Fort George G. Meade MD 20755-6000 1

DoD
R31
9800 Savage Road
Ft. Meade MD 20755-6000 1

DIRNSA
R509
9800 Savage Road
Ft Meade MD 20775 1

Director
NSA/CSS
R08/P & E BLDG
Fort George G. Meade MD 20755-6000 1

DOD Computer Center
C/TIC
9800 Savage Road
Fort George G. Meade MD 20755-6000 1

Rutgers University
Department of Computer Science
Attn: Dr Saul Amarel
Busch Campus
New Brunswick NJ 08903 1

USC-ISI
Attn: Mr Yigal Arens
4676 Admiralty Way
Marina Del Ray, CA 90292 1

University of Southern California 1
Info Sciences Institute
Attn: Dr Robert M. Balzer
4676 Admiralty Way
Marina del Rey CA 90292-6695

TI Central Research Laboratories 1
Computer Science Lab
Attn: Dr Roger Bate, Director
P.O. Box 226015, MS 238
Dallas TX 75266

Northwestern University 1
Institute for Learning Sciences
Attn: Mr Lawrence Birnbaum
1890 Maple Avenue
Evanston IL 60201

ISX 1
Attn: Mr Bruce Bullock
4353 Park Terrace Drive
Westlak Village CA 91361

BBN Laboratories, Inc. 1
Attn: Dr Mark Burstein
10 Mouton Street
Cambridge, MA 02238

Rockwell Int'l Science Center 1
Attn: Mr John Breese
444 High Street
Palo Alto CA 94301

General Electric Company 1
Corporate Research and Development
Attn: Dr Piero P. Bonissone
1 River Road, Building 37-567
Schenectady NY 12345

Carnegie-Mellon University 1
Computer Science Department
Attn: Dr Jaime Carbonell
Schenley Park
Pittsburgh PA 15213

Ohio State University 1
Dept of Comouter & Info Sciences
Attn: Dr B. Chandrasekaran
2036 Neil Avenue
Columbus OH 43210

Brown University
Attn: Dr Eugene Charniak
Box 1910
Providence RI 02912 1

Harvard University
Aiken Computation Lab
Attn: Mr Thomas E. Cheatham
33 Oxford Street
Cambridge, MA 02138 1

Northwestern University
Institute for the Learning Sciences
Attn: Mr Gregory Collins
1890 Maple Avenue
Evanston IL 60201 1

University of Massachusetts
Computer & Info Science Dept
Attn: Dr Paul Cohen
Amherst Massachusetts 01003 1

Oregon State University
Dept of Computer Science
Attn: Dr Bruce D'Ambrosio
Corvallis OR 97331-4602 1

MIT AI lab
Attn: Dr Randall Davis
Room NE43-801A
545 Technology Square
Cambridge MA 02139-1986 1

Brown University
Computer Science Department
Attn: Mr Tom Dean
Box 1910
Providence RI 02912 1

NASA Ames Research
Attn: Mr Mark Drummond
Mailstop 244-17
Moffett Field CA 94035 1

ISX
Attn: Mr Gary Edwards
4353 Park Terrace Drive
Westlake Village CA 91361 1

Tecknowledge, Inc 1
Attn: Dr Lee Erman
1850 Embarcadero
Palo Alto CA 94301

Oakridge National Lab 1
Attn: Mr Robert Edwards
P.O. Box 2008
Building 4500 North, Mailstop 6207
Oakridge TN 37831-6207

DARPA/TTO 1
Attn: Mr. John N. Entzminger
1400 Wilson Boulevard
Arlington VA 22209-2339

Stanford University 1
Knowledge Systems Lab
Attn: Dr Robert Engelmore
701 Welch Road, Bldg C
Palt Alto CA 94304

Jet Propulsion Lab 1
Attn: Mr James Firby
MS 301-440
4800 Oak Grove Drive
Pasadena CA 91109

Carnegie-Mellon University 1
The Robotics Institute
Attn: Mr Mark Fox
5000 Forbes Ave
Pittsourg PA 15213

GTE Labs 1
Attn: Mr W. A. Frawley
40 Sylvan Road
Waltham MA 02254

SRI International 1
AI Center
Attn: Dr Thomas D. Garvey
333 Ravenswood Avenue
Menlo Park CA 94025-3493

Stanford University 1
Attn: Dr Michael R. Genesereth
Heuristic Programming Project
701 Welch Road, Building C
Palo Alto CA 94304

Stanford University
Computer Science Department
Attn: Dr Matt Ginsberg
Stanford CA 94305

1

Kestrel Institute
Attn: Dr. Cordell Green
1801 Page Mill Road
Palo Alto CA 94304

1

University of Chicago
Computer Science Dept/RY 155
Attn: Dr Kris Hammond
1100 E 58th Street
Chicago Illinois 60637

1

Stanford University
Knowledge Systems Lab
Attn: Ms Barbara Hayes-Roth
701 Welch Road Building C
Palo Alto CA 94304

1

Teknowledge, Inc.
Attn: Dr Frederic Hayes-Roth
1850 Embarcadero
Palo Alto CA 94301

1

Cornell University
Computer Science Department
Attn: Dr John Hopcroft
Upson Hall
Ithaca NY 14853

1

Knowledge Systems Lab
Medical Computer Science Group
Attn: Mr Eric Horvitz
MSOB X215
Stanford CA 94305-5479

1

Inference Corporation
Attn: Dr Philip Klahr
5300 West Century Boulevard
Los Angeles CA 90045

1

UCLA
Attn: Dr Richard Korf
Computer Science Department
Los Angeles CA 90024

BBN Systems and Technologies, Corp 1
Attn: Mr Ted Kral
4015 Hancock Street, Suite 101
San Diego CA 92110

George Mason University 1
Info Technology & Engineering
Attn: Dr Paul Lehner
4400 University Drive
Fairfax VA 22033

ADS 1
Attn: Mr Ted Linden
1500 Plymouth St
Mt. View CA 94043

Duke University 1
Computer Science Department
Attn: Dr Donald W. Loveland
Durham NC 27706

University of Chicago 1
Computer Science Dpt. RY 155
Attn: Mr Charles Martin
1100 E 58th Street
Chicago IL 60637

Yale University 1
Dept of Computer Science
Attn: Mr Drew McDermott
51 Prospect Street
New Haven CT 06520

Oakridge National Lab 1
Attn: Mr Bob McLaren
P.O. Box 2008
Building 6011, Mailstop 6370
Oakridge TN 37831-6370

ONR/Code 1133 IS 1
Attn: Mr Alan Meyrowitz
800 North Quincy St
Arlington VA 22217

Intellicorp 1
Attn: Mr Paul Morris
1975 El Camino Real West
Mountain View CA 94040

Stanford University 1
Attn: Dr H. Penny Nii
Heuristic Programming Project
701 Welch Road, Building C
Palo Alto CA 94304

Stanford University 1
Computer Science Department
Attn: Dr Nils J. Nilsson
Margaret Jacks Hall
Stanford CA 94305

University of Chicago 1
Computer Science Dept RY 155
Attn: Chris Owens
1100 E 58th Street
Chicago IL 60637

Babson College 1
Math Department
Attn: Dr Gordon D. Prichett
Babson Park MA 02157-0901

MIT AI Lab 1
Attn: Dr Charles Rich
Room NE43-839
545 Technology Square
Cambridge MA 02139-1936

Bolt, Beranek, and Newman, Inc. 1
Department of AI
Attn: Mr R. Bruce Roberts
10 Moulton Street
Cambridge MA 02238

Teleos Research 1
Attn: Mr Stanley J. Rosenchein
576 Middlefield Rd
Palo Alto CA 94301

Honeywell Systems & Research Center 1
Attn: Mr Robert Schrag
MN 65-2100
3660 Technology Drive
Minneapolis MN 55418

SUNY at Buffalo 1
Computer Science Department
Attn: Dr Stuart C. Shapiro
226 Bell Hall
Buffalo NY 14260

NRL 1
Attn: Dr Randall Shumaker
Code 5510
4555 Overlook Ave, SW
Washington DC 20375-5000

Robotics Institute 1
CMU
Attn: Mr Stephen F. Smith
Schenley Park
Pittsburgh PA 15213

FMC Corporation 1
ATTN: N. S. Sridharan
CTC 1205 Coleman Ave.
Santa Clara CA 95052

Carnegie Mellon University 1
Robotics Institute
School of Computer Science
Attn: Ms Katia Sycara
Pittsburgh PA 15213

AI Applications Institute 1
Attn: Mr Austin Tate
80 Southbridge
Edinburgh, EH1 1HN
United Kingdom

Oakridge National Lab 1
Attn: Mr Bruce Tonn
P.O. Box 2009
Building 4500 North, Mailstop 6207
Oakridge TN 37831-6207

Lockheed AI Center, 90-06/259 1
Lockheed R&D Division
Attn: Dr Steven Vere
3251 Hanover St.
Palo Alto CA 94304-1137

BBN Systems & Technologies Corp 1
Attn: Mr E. Walker
10 Moulton Street
Cambridge MA 02238

Texas Instruments, Inc. 1
AI Lab
Attn: Raj Mall
P.O. Box 655474, M/S 239
Dallas TX 75265

WRDC/TXI 1
Attn: Mr Michael P. Wellman
Bldg 22, Room 5-108
WPAFB OH 45433

SRI International 1
Attn: Mr David Wilkins
333 Ravenswood, EJ 227
Menlo Park CA 94025

MIT AI Lab 1
Attn: Dr Patrick Winston
Room NE43-817
545 Technology Square
Cambridge, MA 02139-1986

DARPA/SYSTO 4
Attn: Lt Col Stephen E. Cross
1400 Wilson Blvd
Arlington VA 22209-2308

The MITRE Corporation 1
Attn: Mr David Day
Burlington Road
Bedford MA 01730

Mr Joseph Fiksel 1
P.O. Box 10119
1350 Embarcadero Rd
Palo Alto CA 94303

Australian AI Institute 1
Attn: Mr Michael Georgeff
1 Grattan St
Carlton, Victoria 3053
Australia

The MITRE Corporation 1
Attn: Mr Mark Nadel
Burlington Rd, MS A047
Bedford MA 01730

Rutgers University 1
Dept of Computer Science
Hill Center, Busch Campus
Attn: Mr Charles Schmidt
New Brunswick NJ 08903

AFOSR/NM 1
Attn: Dr. Abraham Waksman
Bolling AFB DC 20332-6448

University of Rochester 2
Chairman, Dept of Comp Science
Attn: Prof James F. Allen
Computer Studies Building
Rochester NY 14627

Clarkson University 2
Attn: Dr Susan E. Conry
Elect & Computer Eng'g Dept
Potsdam NY 13676-1401

University of Massachusetts 2
COINS Department
Attn: Dr Victor R. Lesser
Lederle Graduate Research Center
Amherst MA 01003-0001

NASA Ames Research Center 1
Attn: Dr Peter E. Friedland
MS 24 4-17
Code RIA
Moffett Field CA 94035-1099

SRI International 1
Attn: Ms Marie A. Bienkowski
333 Ravenswood Ave, EK337
Menlo Park CA 94025

Honeywell Systems & Research Center 1
Attn: Mr Mark S. Boddy
3660 Technology Drive
Minneapolis MN 55418

ESD/AV 1
Brown Building
Hanscom AFB MA 01731-5000

Laboratory for Computer Science 1
Attn: Jon Doyle
Massachusetts Institute of Tech
545 Technology Square
Cambridge MA 02139

UNISYS
Attn: Mr Timothy W. Finin
Ctr for Adv Info Technology
70 East Swedesford Rd
Malvern PA 19355

1

ISX Corporation
Attn: Mr Scott Fouse
501 Marin Street
Suite 214
Thousand Oaks CA 91360

1

University of Massachusetts
Dept of Comp & Info Science
Attn: Dr Edwinna Rissland
Amherst MA 01003

1

The Rand Corporation
Attn: Mr Jeff Rothenberg
1700 Main Street
Santa Monica CA 90406-2138

1

ISX Corporation
Attn: Mr Allen G. Smith
4353 Park Terrace Drive
Westlake Village CA 91361

1

Kestrel Institute
Attn: Mr Douglas Smith
3160 Hillview Ave
Palo Alto CA 94304

1

Rockwell International
Attn: Mr David E. Smith
444 High Street
Palo Alto CA 94301

1

University of Rochester
Attn: Dr. Josh Tenenberg
Computer Science Dept
Wilson Blvd
Rochester NY 14627

1

Standord University
Attn: Gio Wiederhold
Dept of Computer Science
438 Margaret Jacks Hall
Stanford CA 94305-2140

1

Director 1
DARPA/SYSTO
1400 Wilson Blvd
Arlington VA 22209-2308

WRDC/TXI 1
Attn: Mr William Baker
WPAFB OH 45333

U.S. Army Research Office 1
Attn: Dr David Hislop
P.O. Box 1211
Research Triangle NC 27709-2211

U.S. Army Ballistic Rsch Lab 1
Attn: SLCBR-SE-C (M. A. Hirshberg)
Aberdeen Proving Ground MD
21005-5066

Telos Research 1
Attn: Mr David Chapman
572 Middlefield Road
Palo Alto CA 94301

University of Michgan 1
Attn: Mr Edmund H. Durfee
Dept of Elect Eng & CS
1101 Beal Ave
An Arbor MI 48109

MITRE Corporation 1
Attn: Chris Elsaesser
7525 Colshire Deive
McLean VA 22102-3491

University of Maryland 1
Attn: Mr James A. Hendler
Computer Science Dept
UMCP
College Park MD 20742

University of Washington 1
Attn: Mr Steven J. Hanks
Dept of Computer Science & Eng
FR-35
Seattle WA 98195

Anderson Consulting
Attn: Mr Bruce Johnson
100 South Wacker Drive
Chicago IL 60606

1

Teleos Research
Attn: Leslie P. Kaelbing
576 Middlefield Road
Palo Alto CA 94301

1

NASA Ames Research Center
Attn: Pat Langley
M/S 244-17
Moffett Field CA 94035

1

Lockheed R&D Info and CS
Attn: Mr William S. Mark
3251 Hanover Street
90-01 Building 259
Palo Alto CA 94304-1191

1

Carnegie Mellon University
Attn: Raj Reddy
School of Computer Science
Pittsburgh PA 15213

1

Copernican Group
Attn: Mr Earl D. Sacerdoti
787 Melville Ave
Palo Alto CA 94301

1

MITRE Corporation
Attn: Mr Allen Sears
7525 Colshire Drive
M/S Z239
McLean VA 22102

1

Cornell University
Attn: Mr Alberto M. Segre
Dept of Computer Science
Ithaca NY 14853-7501

1

NCR Corporation
Attn: Mr. Robert L. Simoson, Jr.
500 Tech Parkway
Atlanta GA 30313

1

USC-ISI 1
Attn: Dr. Paul Rosenbloom
4676 Admiralty Way
Marina Del Ray CA 90292

BBN Laboratories, Inc. 1
Attn: Mr Richard Estrada
10 Mouton Street
Cambridge MA 02238

BBN Laboratories, Inc. 1
Attn: Mr. Mike Thome
10 Mouton Street
Cambridge MA 02238

Rockwell Int'l Science Center 1
Attn: Mr. Max Henrion
444 High Street
Palo Also CA 94301

SRI International 1
AI Center
Attn: Dr. John Lowrance
333 Ravenswood Avenue
Menlo Park CA 94025-3493

SRI International 1
AI Center
Attn: Dr. Felix Ingrand
333 Ravenswood Avenue
Menlo Park CA 94025-3493

BBN Systems and Technologies Corp 1
Attn: Mr. Mark Fausett
4015 Hancock Street, Suite 101
San Diego CA 92110

Carnegie Mellon University 1
Robotics Institute
Attn: Mr. Donald Kosy
School of Computer Science
Pittsburgh PA 15213

Carnegie Mellon University 1
Robotics Institute
Attn: Mr. Steve Roth
School of Computer Science
Pittsburgh PA 15213

Carnegie Mellon University
Robotics Institute
Attn: Robert Frederking
School of Computer Science
Pittsburgh PA 15213 1

The MITRE Corporation
Attn: Mr. John Woodward
Burlington Rd, MS A047
Bedford MA 01730 1

AFOSR/NM
Attn: Dr. Charles Holland
Bolling AFB DC 20332-6448 1

UNISYS
Attn: Mr. Donald McKay
Ctr for Adv Info Technology
70 East Swedesford Rd
Malvern PA 19353 1

Kestrel Institute
Attnn: Mr. Michael Lowry
3160 Hillview Ave
Palo Alto CA 94304 1

USTRANSCOM (TCJ5-SA)
Scott AFB IL 62225-7001 1

USTRANSCOM (TCJ6-GT)
Scott AFB IL 62225-7001 1

The MITRE Corporation
Attn: Mr. Robert E. Frost
333 Salem Place, Suite 150
Fairview Heights IL 62208 1

The MITRE Corporation
Attn: Mr. Richard Simpson
333 Salem Place, Suite 150
Fairview Heights IL 62208 1

The MITRE Corporation 1
Attn: Mr. Robert McCormick
333 Salem Place, Suite 150
Fairview Heights IL 62208

ESD/AV 1
Hanscom AFB MA 01731

ESD/IC 1
Hanscom AFB MA 01731

ESD/Technical Library 1
Hanscom AFB MA 01731

MITRE Technical Library 1
Hanscom AFB MA 01731

MISSION

OF

ROME LABORATORY

Rome Laboratory plans and executes an interdisciplinary program in research, development, test, and technology transition in support of Air Force Command, Control, Communications and Intelligence (C^3I) activities for all Air Force platforms. It also executes selected acquisition programs in several areas of expertise. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. In addition, Rome Laboratory's technology supports other AFSC Product Divisions, the Air Force user community, and other DOD and non-DOD agencies. Rome Laboratory maintains technical competence and research programs in areas including, but not limited to, communications, command and control, battle management, intelligence information processing, computational sciences and software producibility, wide area surveillance/sensors, signal processing, solid state sciences, photonics, electromagnetic technology, superconductivity, and electronic reliability/maintainability and testability.